On the Complexity of Model Checking Counter Automata

Christoph Haase

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2011

The glory of the human spirit is the single purpose of all science.

Carl Gustav Jacob Jacobi (1804–1851)

Acknowledgements

As stated by John Donne, “no man is an island, entire of itself”. This fact makes life worth living and is the basis that enables one to write a thesis. There is a large number of people and institutions that I would like to thank. This thesis would not have been possible without them.

First of all, I would like to express my deep gratitude to Joël Ouaknine and James Worrell for guiding me through this thesis. The discussions, knowledge, insights and freedom they provided have been of invaluable worth.

I am very grateful to the Engineering and Physical Sciences Research Council (EPSRC) for providing me with a studentship that enabled me to come to Oxford and allowed me to concentrate on my research without any worries.

During my thesis, I have had the fortune to work with a number of collaborators on interesting projects, some of which found their way into this thesis. I am grateful for having worked together with Stefan Göller, Stephan Kreutzer, Jens Lehmann and Matthew Parkinson. Moreover, I would like to thank Byron Cook and Samin Ishtiaq for my stay and the work we did at Microsoft Research Cambridge. I am also grateful for inspiring lab discussions I had with Vidjay D’Silva, Stefan Kiefer, Anthony Widjaja Lin and Georg Weissenbacher.

I am lucky having found so many friends in Oxford, who were also mentors, housemates or inspirators. In particular, I would like to thank Costas Arvanitis, Ana Barros, Miguel Bernabeu, Joseph Chedrawe, Robert Colborn, Matthias Fruth, Christina Fuhr, Robert Gerlach, Theresa Kevorkian, Anneke Kramm, Steve Lew, Erica Marcus, Matthew McGilvray, and Mareike Süße.

St. Catherine’s College has provided me with a rich social and inspiring life outside the lab and the nicest food. I will keep good memories about having rowed for St. Catherine’s College Boat Club.

At the base of all of this is my family. I am grateful to my parents and my sisters Claudia and Susanne for their endless, absolute support and love, and the joy I have when being with them.

Abstract

Theoretical and practical aspects of the verification of infinite-state systems have attracted a lot of interest in the verification community throughout the last 30 years. One goal is to identify classes of infinite-state systems that admit decidable decision problems on the one hand, and which are sufficiently general to model systems, programs or protocols with unbounded data or recursion depth on the other hand.

The first part of this thesis is concerned with the computational complexity of verifying counter automata, which are a fundamental and widely studied class of infinite-state systems. Counter automata consist of a finite-state controller manipulating a finite number of counters ranging over the naturals. A classic result by Minsky states that reachability in counter automata is undecidable already for two counters. One restriction that makes reachability decidable and that this thesis primarily focuses on is the restriction to one counter. A main result of this thesis is to show that reachability in one-counter automata with counter updates encoded in binary is NP-complete, which solves a problem left open by Rosier and Yen in 1986. We also consider parametric one-counter automata, in which counter updates can be parameters ranging over the naturals. Reachability for this class asks whether there are values of the parameters such that a target configuration can be reached from an initial configuration. This problem is also shown to be NP-complete. Subsequently, we establish decidability and complexity results of model checking problems for one-counter automata with and without parameters for specifications written in EF, CTL and LTL.

The second part of this thesis is about the verification of programs with pointers and linked lists in the framework of separation logic. We consider the fragment of separation logic introduced by Berdine, Calcagno and O'Hearn in 2004 and the problem of deciding entailment between formulae of this fragment. We improve the known coNP upper bound and show that this problem can actually be solved in polynomial time. This result is based on a novel approach in which we represent separation logic formulae as graphs and decide entailment between them by checking for the existence of a graph homomorphism. We complement this result by considering various natural extensions of this fragment which make entailment coNP-hard.

Contents

Chapter 1

Introduction

1.1 Background

One of the greatest achievements in human culture in the 20th century may be the formalisation, mechanisation and study of the nature of computation. Although there had been prior attempts to mechanise computation, for example by Leibnitz, who invented a mechanical calculator called the "Stepped Reckoner", or by Babbage's difference engine, no significant insights into the nature of computation had been made until the 20th century. The International Congress of Mathematicians at the Sorbonne in Paris in 1900 can be seen as the starting point of modern developments in the field of computation theory. At the congress, David Hilbert presented ten of his famous 23 problems [59] that were unsolved by the time and that he considered to have a positive influence on the developments of mathematics in the 20th century. The range of the problems considered was broad and, amongst others, concerned with problems in number theory and the foundations of mathematics. Two of Hilbert's problems played a key role in the development of computation theory. Hilbert's second problem deals with the fundamental nature of mathematics as a science and asks to "set up a system of axioms which contains an exact and complete description of the relations subsisting between the elementary ideas of that science [..] [and] to prove that they are not contradictory, that is, that a finite number of logical steps based upon them can never lead to contradictory results." Much to the surprise and disappointment of Hilbert, Gödel showed in 1931 that no such system can exist [47]. Hilbert's tenth problem concerned the solvability of polynomial equations. He asked "to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers." Implicitly, Hilbert asked for an algorithm to solve polynomial equations in the integers, although the concept of an algorithm did not exist by the time. It was in 1970 when Matiyasevich showed [77] that no such algorithm can exist, a solution that Hilbert probably did not consider possible at the time he posed the question.

In the spirit of Hilbert's second problem, researchers in mathematical logic began investigating the consistency of natural logical theories. Notable progress was for example made by Tarski, who showed the consistency of the first-order theory of the reals, and by his student Presburger who proved the consistency of a fragment of number theory, which is known today as Presburger arithmetic [90]. Another result of this type was given by Gödel, who showed the consistency of full first-order logic [46] in 1930 before proving his famous incompleteness theorem [47], which gave an answer to Hilbert's second problem. In this environment, a different branch of research was independently established by Alan Turing and Alonzo Church who independently proposed formal models of computation. Turing's "automatic machine" [104], which is known today as "Turing machine", is an abstract machine that consists of a finite-state controller acting on a tape consisting of infinitely many cells and that has a head which can point to a position on the tape. The contents of the cells of the tape are symbols from a finite alphabet. The behaviour of a Turing machine is specified by a finite list of instructions. When the head reads at the current position of the tape an alphabet symbol in a control state, an instruction determines which symbol to write on the tape and in which direction to move the head by one cell. Some of the control states of a Turing machine are designated accepting control states. Once an accepting control state is entered the machine stops and the computation is finished. Turing claimed that if a function f can naturally be computed, then it can be computed by a Turing machine, *i.e.*, for any such function one can construct a Turing machine such that for any argument to f that is encoded into the tape of a

Turing machine computing f, the Turing machine reaches an accepting control state after a finite number of steps with the value of f encoded into the tape. This claim is known today as the Church-Turing thesis. One can also rephrase this claim and say that any problem that can algorithmically be computed can also be computed by a Turing machine. Turing also showed in [104] that there are functions which are not computable by a Turing machine. In particular, he showed that there does not exist a Turing machine that decides, *i.e.*, answers yes or no, the problem of determining whether a given Turing machine is eventually going to stop for a given input. This problem is known as the halting problem. Problems that cannot be decided by a Turing machine are called undecidable. Another such problem is to decide the truth of statements in arithmetic [27]. Finally, Matiyasevich, building upon the work of Davis, Putnam and Robinson, showed that Hilbert's tenth problem also belongs to the class of undecidable problems [77]. With the advent of physical computers in the 1950s, a further line of research started in the 1960s that investigates the complexity of algorithms. Knowing that a problem can algorithmically be solved does not provide good information about the actual feasibility of an algorithm solving it. The field of computational complexity classifies problems according to their inherent requirements in terms of space and time needed to solve them algorithmically.

The halting problem asks a meta question about computation: given an arbitrary algorithm encoded in a Turing machine, is this algorithm always going to terminate on any input? Nowadays, where computers do not only exist as a theoretical construct but are omnipresent, this is a question of much practical relevance. The question generalises to deciding arbitrary properties of arbitrary algorithms that are implemented as programs. Will the autopilot software of this airplane never get stuck? Is it impossible that the controller of those traffic lights will switch all traffic lights to green at some point? Does the controller of this rocket perform unit conversion correctly? If the answer to any of those questions is "no" lives of humans can be at risk, or huge financial losses may occur as a consequence. Unfortunately, there is no general way of answering those questions for arbitrary programs in an automated fashion: Rice's theorem [93] states that no algorithm can exist that can algorithmically prove any non-trivial specification of an arbitrary program. There are a number of ways out of this dilemma that can help answering questions similar to those above for arbitrary programs, or at least increase the confidence into them. One way is penetrative testing. Setting up and running a number of test cases for a program can reveal errors and thus help to improve the correctness of software. As a downside, it can almost never show the absence of errors and thus its benefit is limited when it comes to software used in safety-critical areas. Nevertheless, testing is the technology that is most commonly used in industry to ensure the functional correctness of software. Another way to overcome Rice's theorem is semi-automated theorem proving. In this approach, a program is translated into a logical language of a theorem prover, and a logic such as Hoare logic [60] is used to reason about the program. A theorem prover is a program that allows the user to write computer-assisted proofs in some general purpose logic. Once a program has been translated into the language of a theorem prover, the user can manually prove the program correct, where the prover assists the user by providing heuristics that can automate some simple proofs. There exist a number of sophisticated theorem provers such as PVS [85], Isabelle/HOL [82] or Coq [42] that have been used to verify programs. A notable recent achievement has been a formal correctness proof of an L4 micro kernel [70] in Isabelle/HOL. Beyond that, proofs of classical mathematical theorems such as Gödels incompleteness theorem or the four-color theorem have been re-proved inside theorem provers [83, 52]. For the verification of programs, the biggest advantage of theorem provers is that any arbitrary program can be proven correct with respect to a specification with less effort than what is needed for paper proofs, if there is a human being capable of doing so. The latter fact is also its biggest disadvantage. Theorem proving requires a high level of expertise and a lot of time. Moreover, if software changes, many proofs have to be reproved. As one of the main properties of software is that it changes, this is one of the reasons why theorem proving has not yet found its way into a mainstream industrial context.

An approach to the verification of programs that lies in between testing and theorem proving, and that this thesis is about, is model checking. Model checking was independently proposed by Clarke and Emerson [31], and by Queille and Sifakis [91]. For an account of the history of model checking, see *e.g.* [30, 3]. Model checking is not exclusively limited to the verification of programs, but can also be used to prove properties of hardware, protocols or other abstract systems. In model checking, in order to prove a program correct with respect to a specification, an abstract model of the program is constructed and the specification is translated into some specification logic. A model checking algorithm can then automatically check if the abstracted program fulfills the specification, *i.e.*, if it is a model of the specification. If this is the case, the algorithm returns "yes", otherwise it provides a counter example that shows how erroneous behaviour of the program occurs. The abstraction step translates a program into a weaker formalism in which Rice's theorem does not apply, which makes automatic verification possible. The biggest advantages of model checking are that it is fully automatic and that it can actually prove the correctness of the abstracted model. On the downside, the abstraction step can abstract away errors that might exist in the original program. Furthermore, the size or state space of the abstracted model can be too big to allow model checking algorithms to be of practical use. Another limitation of the practical application of model checking is that, as in the case of theorem proving, some level of expertise is needed to formulate specifications in a specification logic. A lot of research has been devoted throughout the last 30 years to attenuate those problems. Many techniques and heuristics have been developed that have enabled model checking to find its way from theory into praxis. One application that highlighted the power of model checking occurred in 1996, when Clarke, Khaira and Zhao showed that the floating point division bug of the Intel® Pentium® processor could have been discovered with the technology that was available by the time, and that the correction of the bug provably corrected the bug [29]. Today, a number of plain, general purpose model checking tools exist, but there are also tools devoted to software verification. They have successfully been used in the verification of hard and software in academia and industry. Examples of plain model checkers include SMV, NuSMV [28] or SPIN [62]. Examples of tools devoted to software model checking include BLAST [57], CBMC [32] and SLAM [4]. Over the last 30 years, model checking has evolved as a field of its own inside computer science and has attracted a broad range of research on its theoretical and practical aspects. The importance of the field and the contribution made by Clarke, Emerson and Sifakis has been acknowledged by awarding the 2007 ACM Turing award to the three researchers.

Model checking as proposed by Clarke and Emerson restricts systems to have a finite state space. Moreover, its specification language mainly allows for proving qualitative properties about systems. There are applications domains where this approach is too coarse. Examples include real-time systems, where it is desirable to prove quantitative properties of such systems, for example that a certain action is always performed within a certain amount of time. In order to model such systems, various formalisms have been developed and the model checking approach has been applied to them. One such formalism are timed automata, defined by Alur and Dill in [1]. Starting as a purely theoretical construct, timed automata are nowadays the standard way to model real-time systems. Tools such as the Uppall model checker [9] have been developed and successfully been applied to verifying time-critical properties of real-world systems. Other formalisms that allow for modeling systems in a more fine-grained way and for reasoning about quantitative properties of systems include probabilistic, pushdown or counter automata. One of the main challenges in automatically verifying systems modelled in any of those formalisms is that their state space is infinite, which can quickly lead to undecidability of model checking problems. Even when model checking problems are decidable, a high computational complexity of the model checking algorithms can mean that model checking is practically infeasible. Research on the theoretical side of model checking infinite-state systems focuses on determining the decidability status of such problems and their computational complexity.

1.2 Scope and Contribution of this Thesis

The two main topics of this thesis are theoretical aspects of algorithms for the verification of counter automata and programs with pointers and linked lists. This thesis deals with the two subjects separately in two separated parts. We are going to discuss their relationship at the end of this section.

A counter automaton consist of a finite-state controller which manipulates a finite number of counters ranging over the naturals. At a transition between two control locations, a counter automaton can add and subtract a natural number to and from a counter, respectively, and test a counter for zero. Since the counter values are unbounded, the state space of a counter automaton is infinite. Counter automata were introduced by Minsky [80] as a formal model of computation. They are also known as Minsky machines, counter machines, or counter nets. Minsky showed that two counters are already sufficient for counter automata to be computationally as powerful as Turing machines. Hence, almost all decision problems about counter machines are undecidable. In particular, the most basic verification problem, *reachability*, is undecidable. Given two configurations of a counter automaton consisting of a control location and values of the counters, reachability asks whether there is a path between the two configurations in the transition system induced by the counter automaton. Research has identified several ways of restricting counter automata in order to retain decidability of the reachability problem. Amongst others, this can be achieved by restricting the kinds of allowable tests on the counters (*e.g.* Petri nets [88] which do not allow for zero tests), the types of computations considered (such as reversal boundedness, see *e.g.* [65, 56]), restrictions on the underlying structure of the counter automaton (*e.g.* flatness [34, 73]) and the restriction to only one counter. Decidable classes of counter automata have found applications in a number of areas in verification, for example in modelling resource-bounded processes, numeric data types, programs with lists, recursive or multi-threaded programs, XML query validation, and parameterised hardware verification, see *e.g.* [15, 25, 56, 65, 103]. Moreover, tools such as FLATA [14] exist for manipulating and reasoning about restricted classes

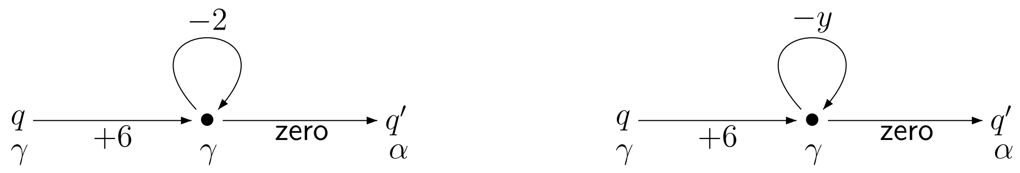

Figure 1.1: An example of a counter automaton and a parametric one-counter automaton.

of counter automata.

This thesis is primarily going to focus on the computational complexity of the verification of one-counter automata, *i.e.*, counter automata with the restriction to only one counter. Such counter automata can for example be used to model programs with one variable, protocols with an unbounded integer storage space, or systems where a transition consumes a resource such as time or money as discussed in [108]. A graphical example of a one-counter automaton is presented on the left-hand side in Figure 1.1. The one-counter automaton consists of three control locations, q, q' and one unnamed location depicted as a bullet, which are labelled with the symbols γ and α. Starting in location q, the one-counter automaton has a transition to the unnamed location, which adds 6 to the counter. Next, it can loop an indefinite number of times subtracting 2 from the counter as long as the counter stays above or equal to 0. Once the counter reaches counter value 0, a transition to the control location q' is enabled. Starting in q, the control location q' with counter value 0 is reachable from q if, and only if, we start with a counter value which is divisible by two. From a complexity perspective, it is important to emphasize that we use the natural *binary* encoding of numbers, unlike a lot of work from the literature, which assumes unary encoding of numbers. One main contribution of this thesis is to show that deciding reachability is an NP-complete problem. A generalisation of one-counter automata are parametric one-counter automata, which are one-counter automata that are equipped with a finite set of parameters. At any transition, a parametric one-counter automaton can add or subtract the value of a parameter. An example of a parametric one-counter automaton is given on the right-hand side of Figure 1.1, which is essentially the same one-counter automaton as on the left-hand side, but in which the loop subtracts the value of the parameter y instead of 2 from the counter. The reachability problem generalises to asking for two given configurations whether there exists an instantiation of the parameters with natural numbers such that one configuration can be reached from another. For example, q' with counter value 0 is reachable from q' with counter value 0 for $y \in \{1, 2, 3, 6\}$. Another result of this thesis is that deciding reachability in parametric one-counter automata is also NP-complete.

Reachability enables us to verify safety properties of systems as it allows for checking whether a designated good or bad state of a system can be reached. In order to verify more complex properties of systems, temporal specification logics are being used in model checking. In this thesis, we deal with the most prominent temporal logics used in verification, the branching-time logics EF and CTL, and the linear-time logic LTL. Those logics allow for specifying properties about the relative order of events on traces in the transition system induced by a one-counter automaton. A trace is the projection onto the labels occurring on a path in the transition system induced by a one-counter automaton. For example, $\gamma\gamma\gamma\gamma\gamma\gamma\alpha$ is the trace of the path that reaches the control location q' with counter value 0 from q with counter value 0 in the one-counter automaton in Figure 1.1. A specification in CTL could for example state that there exists a path whose trace ends in a location labelled with α along which the label γ occurs before. This property is expressed in CTL as follows:

$$\mathsf{E}\ (\gamma\ \mathsf{U}\ \alpha)$$

Here, the E-operator can be read as "there exists a path" and the part in the brackets uses the until operator U and can be read as "γ holds until α holds". This property holds, for example, in the control location q with counter value 0, but not with any counter value that is not divisible by two, as q' cannot be reached from such configurations. From an algorithmic perspective, the decidability of model checking one-counter automata is not trivial. For example, CTL allows to quantify over *all* paths leaving a configuration in the transition system induced by a one-counter automaton, and there can be infinitely many of them. Nevertheless, it has been shown in the literature that model checking EF, CTL and LTL on transition systems generated by one-counter automata is decidable [98, 39]. The contribution of this thesis is to exhaustively determine the computational complexity of those problems.

The model checking problem generalises to parametric one-counter automata. Given a parametric one-counter automaton together with a configuration and a specification in a temporal logic, we aim for determining whether the formula holds in that configuration in all one-counter automata obtained from all valuations of the parameters. We are going to show that this problem becomes undecidable for EF and CTL, but is decidable for LTL.

As a further application of counter automata, the first part of this thesis additionally shows their relationship to the verification of timed systems modeled by timed automata. Timed automata comprise a finite-state controller with a finite number of clocks ranging over the positive reals. One main decision problem for timed automata is reachability. As timed automata are not the main focus of this thesis, we do not give additional details on them here and refer the interested reader to [1, 3]. What we are going to show in this thesis is that reachability problems in timed automata are naturally inter-reducible with reachability problems in bounded counter automata. The latter are counter automata with multiple counters, but each counter is restricted to have some maximum value. The reductions we provide give some interesting insight into the connection between the two formalisms. In particular, we are going to show that reachability in timed automata with two clocks is inter-reducible with reachability in a counter automaton with precisely one bounded counter. The complexity of reachability in two-clock timed automata is one of the major open problems in the theory of timed automata. Although the complexity of reachability in counter automata with exactly one bounded counter remains an open problem of this thesis, it provides a much more simplified formalism that might prove helpful in the future to give an answer to the complexity of reachability in two-clock timed automata.

The second part of this thesis deals with the verification of programs with pointers and linked lists in the framework of separation logic. Separation logic, proposed by Reynolds, O'Hearn, Ishtiaq and Yang [66, 92, 84], is an extension of Hoare logic that allows for reasoning about pointer manipulating programs in an elegant and concise way. It extends the syntax of assertions with predicates describing shapes of memory, which allow for concisely expressing aliasing and disjointness. Full separation logic is very expressive, and most reasoning tasks in this logic are undecidable, which limits its usage for the automatic verification of programs. For that reason, fragments of separation logic with decidable decision problems have been investigated. One such fragment is the one described by Berdine, Calcagno and O'Hearn in [11]. It can be used to reason about structural integrity properties of programs with linked lists and is, for example, the basis of tools like SmallFoot [12]. The fragment described in [11] allows for two predicates for describing the shape of memory:

$$x \mapsto y \qquad\qquad \ell s(y, z)$$

The assertion $x \mapsto y$ can be read as "the memory cell of the stack variable x is allocated on the heap and points to the memory cell of the stack variable y." Moreover, the predicate $\ell s(y, z)$ asserts that there is a possibly empty linked list of arbitrary length from the memory cell of the stack variable y to the memory cell of the stack variable z. The two predicates can then be combined with the star operator in order to describe complex memory shapes. For example, $x \mapsto y * \ell s(y, z)$ describes memory models, which can be decomposed into disjoint parts, one part in which the memory cell of the stack variable x points to y and one in which there is a linked list from y to z. In the part of the heap where $x \mapsto y$ holds, y is not required to be allocated and becomes dangling, which is, informally speaking, the reason why the two parts can "mention" the stack variable y. Additionally, the fragment also allows for asserting conjunctions of equality and inequalities of stack variables, *e.g.*, $x \neq y$ asserts that the stack variables x and y are not equivalent. Since lists have finite but unbounded length, an assertion of the fragment of separation logic that we consider can describe an infinite family of memory models.

The decision problem we investigate in this thesis is entailment between assertions. Given two assertions α and α', entailment is to decide whether α' holds in every memory model in which the assertion α holds. Decidability of this problem was shown in [11] and the authors established a coNP upper bound for this problem. In this thesis, we are going to show that the problem can be solved in polynomial time under a slightly different semantics than the one used in [11]. However, we are going to sketch how this result can be altered in order to give a polynomial-time algorithm for entailment in the semantic model considered in [11]. Our approach is fundamentally different from the one used in [11] and based on graph-theoretic concepts. Moreover, we additionally show that slight adjustments to the syntax of the fragment make entailment coNP-hard.

Although counter automata and separation logic are treated as separate entities in this thesis, they have been shown to be closely related in the literature. Bouajjani *et al.* show in [15] how to verify programs with linked lists with a specification logic similar to the separation logic fragment that we consider via a reduction to verification problems in counter automata. Moreover, Boszga, Iosif and Perarnau have considered in [19] a quantitative version of the separation logic fragment from [11] for which decidability of entailment is shown via a translation into bisimilar counter automata. Both [15] and [11] do not give any complexity bounds of their approaches, and there is no compelling way to obtain any bounds close to those that are provided in this thesis since their constructions lead to an exponential blowup.

In summary, this thesis makes the following novel contributions:

- It shows that reachability in one-counter automata with updates encoded in binary is NP-complete. This solves a problem left open in [95] about the complexity of boundedness in Petri nets with one place and zero tests.
- It shows that reachability in parametric one-counter automata is NP-complete.
- It shows that reachability problems in timed and bounded counter automata are inter-reducible and establishes a dichotomy of those reductions with respect to the resources of the timed and bounded counter automata.
- It exhaustively determines the computational complexity of model checking one-counter automata with updates encoded in binary and parametric one-counter automata with specifications given in the specification logics EF, CTL and LTL.

- It shows that entailment in a fragment of separation logic with pointers and linked lists can be computed in polynomial time, answering a question left open in [11].

1.3 Structure and Style of this Thesis

The organisation of this thesis is as follows. Chapter 2 introduces basic notation and concepts that this thesis builds upon. The aim of this chapter is to give all definitions and results from the literature relevant to this thesis and covers areas such as computational complexity, automata theory, logic and arithmetic theories. Definitions relevant to a specific chapter are introduced in the respective chapter. After Chapter 2, the thesis is broken into two independent parts.

The first part deals with counter automata. Chapter 3 establishes a natural connection between reachability problems in bounded counter automata and timed automata and shows that both problems are inter-reducible. Chapter 4 then deals with the computational complexity of reachability problems in counter automata with and without parameters. Finally, Chapter 5 is devoted to the computational complexity of model checking one-counter and parametric one-counter automata.

The second part, Chapter 6 of this thesis, is about the verification of programs with pointers and linked lists in the framework of separation logic. It is shown there that entailment in the separation logic fragment described in [11] is computable in polynomial time, and this result is complemented by showing that entailment in natural extensions of this fragment becomes intractable.

Each chapter closes with a discussion about the results obtained, how they fit into the literature and possible future work. Since the two parts of this thesis and the chapters therein are all separate entities, we omit a chapter concluding the whole thesis as it would just consist of a repetition of this introductory chapter and the conclusions of the other chapters.

As discussed previously in this chapter, model checking is an established field and has produced a rich body of literature. Although this thesis is self contained, giving a full account on the history, motivation and intuitive meaning of all concepts, definitions and results used from the literature would go beyond the scope of this thesis. The reader of this thesis is expected to have some level of familiarity with concepts and standard results from theoretical computer science. In particular, this thesis requires some knowledge of standard definitions and results in computational complexity, automata theory, infinite-state system verification, model checking, graph theory and arithmetic theories, in particular Presburger arithmetic. Additionally, Chapter 6 requires some level of familiarity with separation logic. The papers by Reynolds [92] and Berdine *et al.* [11] give a good introduction to the fragment considered in this thesis. However, no knowledge of separation logic is required for any of the other chapters.

Literature used by the author which inspired some of the definitions used in this thesis amongst others include Sipser's book on computational complexity [100]; the books by Clarke *et al.* [33] and Baier and Katoen on model checking [3]; Schnoebelen's paper on the complexity of temporal model checking [97]; and Smorynski's book on logical number theory [102].

1.4 Related Work

This section discusses some work from the literature that is related to the main topics of this thesis. Additional related work will be discussed in the respective chapters.

The relationship between timed automata and counter automata.

The classical undecidability proof of universality of timed automata by Alur and Dill [1] proceeds via a reduction from the reachability problem of two-counter automata, which shows a connection between the two formalisms. Recent work [44] by Figueira, Hofman and Lasota establishes a relationship between timed automata and register automata. The latter are somewhat similar to counter automata, but still of a different nature, which means that their result is incomparable to the result provided in this thesis. Register automata include registers that can store data values for later comparison. The paper [44] provides an exponential-time algorithm which computes a register automaton corresponding to a timed automaton and *vice versa.* Runs in the timed automaton can then be simulated by the corresponding register automaton and *vice versa.*

Reachability in one-counter automata.

Reachability in counter automata was first investigated by Minsky [80] who showed that this problem is undecidable in the presence of at least two counters. As discussed in Section 1.2, there is a large body of work on various restrictions on counter automata for which reachability becomes decidable. The problem of the complexity of reachability in one-counter automata with updates encoded in binary has first been mentioned by Rosier and Yen in a paper on the complexity of the boundedness problems for Petri nets, where this problem is left open [95]. Related work on reachability in one-counter automata has also been conducted by Lafourcade *et al.* [71] who show that reachability in one-counter automata with updates encoded in unary is decidable in NL. Based on their work, Demri and Gascon show in [39] that Büchi reachability in such counter automata with additional sign tests on the counter is NL-complete. As in this thesis, deep inspection of the structure of runs in one-counter automata is also the basis of work on the complexity of bisimulation between one-counter automata with updates encoded in unary [13].

Reachability in parametric one-counter automata.

Work closely related to reachability in parametric one-counter automata is that of Ibarra *et al.* [65], which shows decidability of reachability for a subset of the class of deterministic parametric one-counter automata with sign tests. The decidability of reachability over the whole class of such automata is stated as an open problem in [65]. Note that although in this thesis parametric one-counter automata do not allow negative counter values and sign tests, they allow non-determinism. Thus, the results of this thesis are incomparable to those from [65]. Bozga, Iosif and Lakhnech [18] show decidability of the reachability problem for flat parametric counter automata with a single loop, by a reduction to a decidable problem concerning quadratic Diophantine equations. Such systems of equations also feature in the work of Ibarra and Dang [64]. They exhibit a connection between a decidable class of quadratic Diophantine equations and a class of counter automata with reversal-bounded counters.

Model Checking One-Counter Automata

The literature contains a wide range of work on model checking problems of one-counter automata with updates in unary. Serre [98] establishes a PSPACE upper bound for model checking the modal μ-calculus on transition systems generated by such counter automata. A PSPACE lower bound was later established by Göller and Lohrey [50], who showed that model checking CTL on one-counter automata with updates encoded in unary is PSPACE-complete. The proof of the lower bound of CTL model checking on one-counter automata with updates in binary given in this thesis is inspired by the proof of the lower bound in [50]. Model checking EF on one-counter automata with updates in unary was first considered by Jančar *et al.* [68], who showed that this problem is DP-hard. The precise complexity was later settled by Göller, Mayr and To in [51], where it is shown that the problem is P^{NP}-complete. Model checking LTL on one-counter automata with updates in unary has been investigated by Demri and Gascon in [39] and shown to be PSPACE-complete. Also related is the work by Walukiewicz, who studied in [107] model checking EF and CTL on transition systems generated by pushdown automata and showed that the problem is PSPACE-respectively EXPTIME-complete. A topic slightly more remotely related to this thesis is model checking Freeze LTL on one-counter automata, which has been investigated by Demri *et al.* [40, 41]. Freeze LTL extends LTL by the ability to store a counter value and to later test it against the current counter value.

Programs with linked lists in the framework of separation logic.

The fragment of separation logic this thesis deals with has been introduced by Berdine, Calcagno and O'Hearn in [11], where it was shown that entailment in this fragment is decidable and in coNP. Bozga, Iosif and Perarnau considered in [19] a quantitative version of this logic, which additionally allows for reasoning about properties of lengths of lists. Their decidability result also yields a decision procedure for the fragment considered in [11]. However, there are no complexity bounds given in [19] on the algorithm, which is of at least exponential running time. Bansal, Brochenin and Lozes also consider in [5] an extension of the fragment of separation logic that this thesis deals with, which additionally allows for comparing consecutive data in a list. Finally, Navarro Pérez and Rybalchenko developed in [81] a decision procedure for an extension of the fragment from [11] based on superposition. However, the authors do not give complexity bounds of their approach.

1.5 Joint Work

The results presented in this thesis are partly based on peer-reviewed publications that have been co-authored with a number of collaborators. The content of those publications is a result of a number of discussions between the author and his collaborators, in person or via email. Some of the results in Chapter 4 have been published in the proceedings of CONCUR'09 [54]. The paper was co-authored by Stephan Kreutzer, Joël Ouaknine and James Worrell. Results from Chapter 5 have been published in the proceedings of ICALP'10 [48] and FoSSaCS'12 [49]. Both papers have been co-authored by Stefan Göller, Joël Ouaknine and James Worrell. The results from Chapter 6 have been published in the proceedings of CONCUR'11 [35], the paper was co-authored by Byron Cook, Joël Ouaknine, Matthew Parkinson and James Worrell.

Chapter 2

Preliminaries

In this chapter, we are going to introduce general notation and concepts and recall some results from the literature that this thesis builds upon. The first section provides general notation on integers and language theory and gives a brief account on some results from number theory. The next section then introduces transition systems. Section 2.3 deals with formal languages. In particular, we are going to define regular languages and complexity classes. Subsequently, we are going to introduce formal models of finite and infinite state systems, namely finite-state machines, counter automata and timed automata. The last section deals with theories of integer arithmetic with a particular focus on Presburger arithmetic.

2.1 General Notation

By $\mathbb{R}$ we denote the set of *reals*, by $\mathbb{Q}$ the set of *rationals*, by $\mathbb{Z}$ the set of *integers*, and by $\mathbb{N} \stackrel{\text{def}}{=} \{n \geq 0 : n \in \mathbb{Z}\}$ the set of *naturals*. We denote by $\mathbb{R}_{\geq 0} \stackrel{\text{def}}{=} \{r \in \mathbb{R} : r \geq 0\}$ the set of positive reals and by $\mathbb{N}_{>0} \stackrel{\text{def}}{=} \mathbb{N} \setminus \{0\}$ the set of naturals strictly greater than zero. For any $z \in \mathbb{Z}$, $|z|$ is the *absolute value* of z. Given a set M, we denote by 2^M the *power set of* M. For a given a set $M \subseteq \mathbb{Z}$ with a maximum or minimum element, we denote by $\max M$ the *maximum element* of M and by $\min M$ the *minimal element* of M. The size of a finite set M is denoted by $\#M$. Given a relation $R \subseteq M \times N$ and $m \in M$, $R(m) \stackrel{\text{def}}{=} \{n \in N : (m, n) \in R\}$. We write $M \subseteq_{\text{fin}} N$ to say that the set M is a finite subset of N. Moreover, we write $f : M \rightharpoonup_{\text{fin}} N$ to indicate that f is a partial function from M to N with a finite domain. For $M \subseteq \mathbb{R}$, $q \in \mathbb{Q}$ and $r \in \mathbb{R}$, we define $qM \stackrel{\text{def}}{=} \{qm : m \in M\}$ and $M + r \stackrel{\text{def}}{=} \{m + r : m \in M\}$. For each $i, j \in \mathbb{Z}$, we define $[i, j] \stackrel{\text{def}}{=} \{z \in \mathbb{Z} : i \leq z \leq j\}$ and $[i] \stackrel{\text{def}}{=} [1, i]$. Given $r \in \mathbb{R}$, we define the *floor function* $\lfloor r \rfloor \stackrel{\text{def}}{=} \max\{z \in \mathbb{Z} : z \leq r\}$ and the *ceiling function* $\lceil r \rceil \stackrel{\text{def}}{=} \min\{z \in \mathbb{Z} : z \geq r\}$. For any $n \in \mathbb{N}$, we define $\lg n \stackrel{\text{def}}{=} \min\{i \in \mathbb{N} : 2^i \geq n\}$. Throughout this thesis, if not stated otherwise, we assume *binary encoding of numbers*, *i.e.*, the size of any $z \in \mathbb{Z}$ is $\lg |z|$. Given functions $f, g : \mathbb{N} \to \mathbb{N}$, we write $f = O(g)$ if there exist $m, n_0 \in \mathbb{N}$ such that $f(n) \leq mg(n)$ for all $n > n_0$. Given a function $f : M \to N$, we write $f[m_0 \mapsto n_0]$ to denote the function

$$f[m_0 \mapsto n_0] \stackrel{\text{def}}{=} m \mapsto \begin{cases} n_0 & \text{if } m = m_0 \\ f(m) & \text{otherwise.} \end{cases}$$

Let Σ be a set of *letters* forming an *alphabet*. A *finite word w over Σ of length* $n \in \mathbb{N}$ is a function $w : [n] \to \Sigma$, where the empty function is called the *empty word* and denoted by ϵ. We write $|w|$ to denote the length of w. Alternatively, we represent finite words as finite sequences of letters from Σ, *i.e.*, write $w = \sigma_1 \sigma_2 \ldots \sigma_n$. An *infinite word w over* Σ is a function $w : \mathbb{N}_{>0} \to \Sigma$, and its length is $|w| \stackrel{\text{def}}{=} \omega$. Given a word w, for any $i \in [0, |w|]$ we denote by $w^{(i)} : [|w| - i] \to \Sigma$ the *suffix of w starting at position i*, which is defined as $w^{(i)}(j) \stackrel{\text{def}}{=} w(i + j)$ for all $j \in [|w| - i]$. Given a finite word w_1 and a possibly infinite word w_2 over Σ, the *concatenation* $w_1 \cdot w_2$ of w_1 and w_2 is the word $w : [|w_1| + |w_2|] \to \Sigma$, where $w(i) \stackrel{\text{def}}{=} w_1(i)$ for all $i \in [w_1]$ and $w(j) \stackrel{\text{def}}{=} w_2(j - |w_1|)$ for all $j \in [|w_1| + 1, |w_1| + |w_2|]$. For any finite word w over Σ and $n \in \mathbb{N}$, we inductively define w^n as $w^0 \stackrel{\text{def}}{=} \epsilon$ and $w^i \stackrel{\text{def}}{=} w^{i-1} \cdot w$ for all $i > 0$. Moreover, we inductively define $\Sigma^0 \stackrel{\text{def}}{=} \{\epsilon\}$ and $\Sigma^i \stackrel{\text{def}}{=} \{w \cdot \sigma : w \in \Sigma^{i-1}, \sigma \in \Sigma\}$. The set Σ^* of *all finite words* over Σ is defined as $\Sigma^* \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} \Sigma^i$. The set of all infinite words is denoted by $\Sigma^\omega \stackrel{\text{def}}{=} \{w : w \text{ is a function } \mathbb{N}_{>0} \to \Sigma\}$. A subset $L \subseteq \Sigma^*$ or $L \subseteq \Sigma^\omega$ is called a *language*. Given languages $L_1, L_2 \subseteq \Sigma^*$, we denote by $L_1 \cdot L_2 \stackrel{\text{def}}{=} \{w_1 \cdot w_2 : w_i \in L_i, i \in \{1, 2\}\}$ the concatenation of L_1 with L_2. For languages consisting of a single word w, we abuse notation and sometimes write $w \cdot L$ instead of $\{w\} \cdot L$.

Given $x, y \in \mathbb{Z}$, *integer division* is defined as $x \operatorname{div} y \stackrel{\text{def}}{=} \lfloor x/y \rfloor$. We write $x|y$

if x *divides* y, *i.e.*, if there exists a $k \in \mathbb{Z}$ such that $y = kx$. Let $z \in \mathbb{Z}$ and $n \in \mathbb{N}_{>0}$, the *congruence class of z modulo n* is the set $\overline{z}_n \overset{\text{def}}{=} \{z + in : i \in \mathbb{Z}\}$. Each $n \in \mathbb{N}_{>0}$ yields n congruence classes $\mathbb{Z}/\mathbb{Z}_n \overset{\text{def}}{=} \{\overline{0}_n, \ldots, \overline{n-1}_n\}$. For $x, y \in \mathbb{Z}$, we write $x \equiv y \mod n$ if $\overline{x}_n = \overline{y}_n$. The *greatest common divisor* of $x, y \in \mathbb{Z}$ is defined as $\gcd(x, y) \overset{\text{def}}{=} \max\{n \in \mathbb{N} : n|x \textit{ and } n|y\}$ and the *least common divisor* is $\operatorname{lcm}(x, y) \overset{\text{def}}{=} xy/\gcd(x, y)$. Given a non-empty finite set $M = \{z_1, \ldots, z_n\} \subseteq \mathbb{Z}$, $\gcd M \overset{\text{def}}{=} \gcd(z_1, \gcd(z_2, \ldots))$. Likewise, $\operatorname{lcm}(M) \overset{\text{def}}{=} \operatorname{lcm}(z_1, \operatorname{lcm}(z_2, \ldots))$. A natural number $p > 1$ is a *prime number* if $n \not| p$ for all $n \in [2, p-1]$. Let $\pi(n)$ be the prime-counting function that counts the number of primes less or equal to $n \in \mathbb{N}$. It follows from the prime number theorem that for all $n \in \mathbb{N}$, $\pi(n) \sim n/(\lg n)$, *i.e.*, $\lim_{n\to\infty}(\pi(n) \lg n/n) = 1$. The prime number theorem guarantees that the set of all natural numbers up to a fixed size asymptotically contains an exponential number of prime numbers, a fact expolited when proving lower bounds in Chapter 5. In order to obtain a fixed bound on $\pi(n)$, one can, for example, use a result by Rosser [96], which states that for all $n \geq 55$, $n/(\ln n + 2) < \pi(n) < n/(\ln n - 4)$.

Given $i, n \in \mathbb{N}$, $\operatorname{bit}_i(n) \in \{0, 1\}$ denotes the i-th least significant bit of the binary representation of n, *i.e.*, $n = \sum_{i\in\mathbb{N}} 2^i \operatorname{bit}_i(n)$. Moreover, we also represent natural numbers as bit strings over the alphabet $\{0, 1\}$. The *binary representation of n aligned to $m \geq \lg n$* is the word $w \in \{0, 1\}^m$ such that $n = \sum_{i\in[m]} 2^{i-1} w(i)$ and denoted by $\operatorname{bin}_m(n)$. Note that we use little-endian representation when representing numbers in binary as bit strings over $\{0, 1\}$. Given $i < j \in \mathbb{N}$ and $n \in \mathbb{N}$, we denote by $n[i, j]$ the bit string $\operatorname{bit}_i(n) \operatorname{bit}_{i+1}(n) \ldots \operatorname{bit}_{j-1}(n)$. Given a bit string $w \in \{0, 1\}^*$, we denote by $(w)_2$ the natural number $n \overset{\text{def}}{=} \sum_{i\in[|w|]} 2^{i-1} w(i)$. When working in different bases, given a basis $m > 1$ and $i \in \mathbb{N}$, $\operatorname{dig}_i(n)$ denotes the i-th *digit* of n in base m. For example, in base 2 we have $\operatorname{dig}_2(01001) = 1$, in base 10 we have $\operatorname{dig}_2(1312) = 1$, and in base 16 we have $\operatorname{dig}_2(\mathrm{BF0D}) = 0$.

2.2 Transition Systems

As stated in the introduction in the first chapter, one main aspect of this thesis is to study the computational complexity of model checking problems for *transition systems* generated by certain classes of automata.

Definition 1 A *transition system* is a tuple $T = (S, \rightarrow)$, where S is the set of *states* and $\rightarrow \subseteq S \times S$ is the *transition relation.* A *labeled transition system* $T = (S, \rightarrow, \Lambda, \lambda)$ additionally comprises of a finite set Λ of *labels* and a *labelling function* $\lambda : S \rightarrow 2^{\Lambda}$ that assigns a set of labels to each state.

We prefer to use infix notation and write $s \rightarrow s'$ whenever $(s, s') \in \rightarrow$. By $\rightarrow^*$ we denote the reflexive transitive closure of $\rightarrow$. A *finite run ϱ in T of length n* is a finite word $\varrho = s_1 \ldots s_{n+1}$ such that $s_i \rightarrow s_{i+1}$ for $i \in [n]$. We write $\varrho : s \rightarrow^* s'$ if $s_1 = s$ and $s_{n+1} = s'$. An *infinite run in T* is an infinite word $\varrho : \mathbb{N}_{>0} \rightarrow S$ such that $\varrho(i) \rightarrow \varrho(i+1)$ for all $i > 0$. Given a finite run ϱ_1 of length n and a run ϱ_2 such that $\varrho_1(n+1) = \varrho_2(1)$, the *composition of ϱ_1 and ϱ_2* is the run $\varrho \overset{\text{def}}{=} \varrho_1 \cdot {\varrho_2}^{(1)}$, *i.e.*, the run obtained from concatenating ϱ_1 and ϱ_2 without the duplicate first state of ϱ_2. For any subset $S' \subseteq S$, an infinite run ϱ is a *Büchi run in S'* if for any $i \in \mathbb{N}$ there is $j > i$ such that $\varrho(j) \in S'$, *i.e.*, states from S' occur infinitely often along ϱ. In a labeled transition system, the *trace τ of a run ϱ* of length n is the word $\tau : [n+1] \rightarrow 2^{\Lambda}$ which maps every state along ϱ to its label, *i.e.*, $\tau(i) \overset{\text{def}}{=} \lambda(\varrho(i))$ for all $i \in [n+1]$.

One central question that we consider in this thesis is to decide *reachability* in transition systems.

<u>REACHABILITY IN TRANSITION SYSTEMS</u>

INPUT: A transition system $T = (S, \rightarrow)$ and $s, s' \in S$.

QUESTION: Does $s \rightarrow^* s'$?

Given two labelled transition systems $T_i = (S_i, \rightarrow_i, \Lambda, \lambda_i), i \in \{1, 2\}$, the *product* $T = T_1 \times T_2$ of T_1 and T_2 is the transition system $T = (S, \rightarrow, \Lambda, \lambda)$, where

- $S \overset{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 : \lambda_1(s_1) = \lambda_2(s_2)\}$;

- $\rightarrow \overset{\text{def}}{=} \{((s_1, s_2), (s'_1, s'_2)) : (s_1, s_2), (s'_1, s'_2) \in S, s_i \rightarrow_i s'_i, i \in \{1, 2\}\}$; and
- $\lambda \overset{\text{def}}{=} (s_1, s_2) \mapsto \lambda_1(s_1)$.

Observe that for any trace τ, we have that τ is the trace of a run ϱ in T if, and only if, τ is a trace of runs ϱ_1 and ϱ_2 in T_1 respectively T_2.

2.3 Formal Languages and Computational Complexity

In this section we recall some definitions and results from formal language and complexity theory. Sipser's book [100] provides the basis for most of our definitions.

2.3.1 Regular Languages

Regular languages are languages consisting of finite words that can be accepted by a finite automaton.

Definition 2 Let Σ be a finite alphabet. A *deterministic finite automaton (DFA)* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$, where Q is a finite set of *control locations* , $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, and $\delta : Q \times \Sigma \rightarrow Q$ is the *transition function*.

Let $w = \sigma_1 \dots \sigma_n$ be a finite word over Σ. Then $\mathcal{A}$ *accepts* w if there exists a word $r = r_0 \dots r_n$ over Q such that

- $r_0 = q_0$
- $\delta(r_i, \sigma_{i+1}) = r_{i+1}$ for $i \in [0, n-1]$; and
- $r_n \in F$.

The *language* $L(\mathcal{A})$ *accepted by* $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$, *i.e.*,

$$L(\mathcal{A}) \overset{\text{def}}{=} \{w \in \Sigma^* : \mathcal{A} \text{ accepts } w\}.$$

The class REG of *regular languages* over Σ is defined as

$$\mathsf{REG} \overset{\text{def}}{=} \{L \subseteq \Sigma^* : L \text{ is accepted by some DFA } \mathcal{A}\}.$$

2.3.2 Turing Machines

Turing machines (TM) provide a formal model of a computer. The literature contains a large body of different, though equivalent, definitions of Turing machines. Mainly for the purpose of technical convenience, in this thesis a Turing machine operates on one input and one working tape over the alphabet $\Sigma \overset{\text{def}}{=} \{0, 1, \triangleright, \triangleleft\}$. Here, $\triangleright$ and $\triangleleft$ are special symbols serving as delimiters to mark the beginning of the input and working tape and the end of the input tape, respectively. The area to the right of the $\triangleright$ delimiter of the working tape is assumed to be initially filled with zeros. Let $\Upsilon \overset{\text{def}}{=} \{-1, 0, +1\}$ denote the set of *head directions*, where -1 indicates that a head moves to the left, 0 that it stays at its current position, and $+1$ that it moves to the right.

Definition 3 Let $\Sigma \overset{\text{def}}{=} \{0, 1, \triangleright, \triangleleft\}$ be an alphabet. A *Turing machine* is a tuple $\mathcal{M} = (Q, \Sigma, q_0, A, R, \Delta)$, where Q is a finite set of *control locations*, q_0 is the *initial location*, $A \subseteq S$ is the set of *accepting locations*, $R \subseteq S \setminus A$ is the set of *rejecting locations*, and $\Delta \subseteq Q \times \Sigma^2 \times Q \times \Sigma \times \Upsilon^2$ is the *transition relation*.

In order to capture the intuitive meaning of the delimiters and that of the accepting and rejecting locations, we impose the following restrictions on the transition relation: for any $(s, \sigma_{\mathsf{i}}, \sigma_{\mathsf{w}}, s', \sigma'_{\mathsf{w}}, d_{\mathsf{i}}, d_{\mathsf{w}}) \in \Delta$

- if $\sigma_{\mathsf{i}} = \triangleright$ then $d_{\mathsf{i}} \neq -1$, and if $\sigma_{\mathsf{i}} = \triangleleft$ then $d_{\mathsf{i}} \neq +1$;
- if $\sigma_{\mathsf{w}} = \triangleright$ then $\sigma'_{\mathsf{w}} = \triangleright$ and $d_{\mathsf{w}} \neq -1$; and
- $s \notin A \cup R$.

The first constraint ensures that the input head does not move beyond the delimiters, the second that the working head does neither move beyond nor overwrite the left delimiter, and the third that the Turing machine stops whenever it reaches an accepting or rejecting location. Moreover, we require that for any $q \in Q \setminus (A \cup R)$ and $\sigma_{\mathsf{i}}, \sigma_{\mathsf{w}} \in \Sigma$, there exists one tuple $(q, \sigma_{\mathsf{i}}, \sigma_{\mathsf{w}}, q', \sigma'_{\mathsf{w}}, d_{\mathsf{i}}, d_{\mathsf{w}}) \in \Delta$ for some $q', \sigma'_{\mathsf{w}}, d_{\mathsf{i}}, d_{\mathsf{w}}$. We call $\mathcal{M}$ *deterministic* if there is exactly one such tuple, otherwise $\mathcal{M}$ is called *non-deterministic*. A *configuration* C of $\mathcal{M}$ is a tuple $(q, h_{\mathsf{i}}, h_{\mathsf{w}}, w_{\mathsf{i}}, w_{\mathsf{w}})$, where $q \in Q$ is the current state of $\mathcal{M}$; $w_{\mathsf{i}} \in \triangleright \cdot \{0,1\}^* \cdot \triangleleft, w_{\mathsf{w}} \in \triangleright \cdot \{0,1\}^*$ are the contents of the input respectively working tape; and $h_{\mathsf{i}} \in [|w_{\mathsf{i}}|], h_{\mathsf{w}} \in [|w_{\mathsf{w}}|]$ are the current positions of the heads on the input respectively working tape. Denote by $C(\mathcal{M})$ the set of all configurations of $\mathcal{M}$. Then $\mathcal{M}$ induces the transition system $T(\mathcal{M}) = (C(\mathcal{M}), \to_{\mathcal{M}})$ where for $C = (q, h_{\mathsf{i}}, h_{\mathsf{w}}, w_{\mathsf{i}}, w_{\mathsf{w}})$ and $C' = (q', h'_{\mathsf{i}}, h'_{\mathsf{w}}, w'_{\mathsf{i}}, w'_{\mathsf{w}})$, we have $C \to_{\mathcal{M}} C'$ if, and only if, there exists $(q, \sigma_{\mathsf{i}}, \sigma_{\mathsf{w}}, q', \sigma'_{\mathsf{w}}, d_{\mathsf{i}}, d_{\mathsf{w}}) \in \Delta$ such that

- $w_{\mathsf{i}}(h_{\mathsf{i}}) = \sigma_{\mathsf{i}}$, $w_{\mathsf{w}}(h_{\mathsf{w}}) = \sigma_{\mathsf{w}}$
- $h'_{\mathsf{i}} = h_{\mathsf{i}} + d_{\mathsf{i}}$, $h'_{\mathsf{w}} = h_{\mathsf{w}} + d_{\mathsf{w}}$
- $w'_{\mathsf{i}} = w_{\mathsf{i}}$
- $w'_{\mathsf{w}} = w_{\mathsf{w}}[h_{\mathsf{w}} \mapsto \sigma'_{\mathsf{w}}]$ if $h_{\mathsf{w}} + d_{\mathsf{w}} < |w_{\mathsf{w}}|$, and $w'_{\mathsf{w}} = w_{\mathsf{w}}[h_{\mathsf{w}} \mapsto \sigma'_{\mathsf{w}}] \cdot 0$ otherwise.

The latter condition implies that the working tape can be seen to be initially filled with zeros. Also notice that the above restrictions on Δ ensure that $T(\mathcal{M})$ is well-defined.

Let $\Sigma_{\mathsf{i}} \overset{\text{def}}{=} \{0,1\}$ be the *input alphabet* and $w \in \Sigma_{\mathsf{i}}^*$ an input word to $\mathcal{M}$. The configuration $C = (q, h_{\mathsf{i}}, h_{\mathsf{w}}, w_{\mathsf{i}}, w_{\mathsf{w}})$ is called the *initial configuration* if $q = q_0$, $w_{\mathsf{i}} = \triangleright \cdot w \cdot \triangleleft$, $w_{\mathsf{w}} = \triangleright \cdot 0$, and $h_i = h_w = 2$. We call C an *accepting configuration* if $q \in A$, and a *rejecting configuration* if $q \in R$. A configuration is *terminating* if it is accepting or rejecting. Given an input w, a run $r : C \to_{\mathcal{M}}^* C'$ is a *computation of* $\mathcal{M}$ *on* w if C is the initial configuration and C' is a terminating configuration. The length of a computation r of $\mathcal{M}$ is $|r|$. We say $\mathcal{M}$ *accepts* w if there exists a computation ending in an accepting configuration. The *language accepted by* $\mathcal{M}$ is

$$L(\mathcal{M}) \overset{\text{def}}{=} \{w \in \Sigma_{\mathsf{i}}^* : \mathcal{M} \text{ accepts } w\}.$$

Note that in general the input alphabet may contain an arbitrary number of symbols, since an arbitrary input alphabet can be encoded into Σ_i by using standard constructions. The *characteristic function* $\chi_\mathcal{M} : \Sigma_\mathsf{i}^* \to \{0,1\}$ of $\mathcal{M}$ is defined as $\chi_\mathcal{M}(w) \overset{\text{def}}{=} 1$ if $w \in L(\mathcal{M})$, and $\chi_\mathcal{M}(w) \overset{\text{def}}{=} 0$ otherwise. We call $\mathcal{M}$ a *decider* if for any $w \in \Sigma_\mathsf{i}^*$ there is no infinite run in $T(\mathcal{M})$ starting in the initial configuration.

2.4 Computability and Computational Complexity

Let Σ be an alphabet. The set of *recursively enumerable languages*, also known as the set of languages in the first level of the arithmetic hierarchy, is defined as

$$\Sigma_1^0 \overset{\text{def}}{=} \{L \subseteq \Sigma^* : L = L(\mathcal{M}) \text{ for some TM } \mathcal{M}\}.$$

A language $L \subseteq \Sigma^*$ is *decidable* if $L \in \Sigma_1^0$ and $\Sigma^* \setminus L \in \Sigma_1^0$, or alternatively if $L = L(\mathcal{M})$ for some decider $\mathcal{M}$. A language is called *undecidable* if it is not decidable.

We now define time and space complexity classes. In what follows, all Turing machines we consider are deciders. The *running time* of a Turing machine $\mathcal{M}$ is a function $f : \mathbb{N} \to \mathbb{N}$ such that for any input word $w \in \Sigma^n$, the length of any computation of $\mathcal{M}$ on w is at most $f(n)$. We call such a Turing machine an *$f(n)$-time Turing machine.* In particular, we call $\mathcal{M}$ *a deterministic polynomial-time Turing machine* if $\mathcal{M}$ is a deterministic $f(n)$-time Turing machine for some polynomial f. The *space complexity* of $\mathcal{M}$ is the function $f : \mathbb{N} \to \mathbb{N}$ such that for any input word $w \in \Sigma^n$, the position of the working head on any computation of $\mathcal{M}$ on w is at most $f(n)$. We call such a Turing machine a *$f(n)$-space Turing machine.*

The classes of languages decided by time- and space-bounded Turing machines

$$
\begin{aligned}
\mathsf{P} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{DTIME}\left(n^i\right) & \mathsf{NP} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{NTIME}\left(n^i\right) \\
\mathsf{EXPTIME} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{DTIME}\left(2^{n^i}\right) & \mathsf{NEXPTIME} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{NTIME}\left(2^{n^i}\right) \\
\mathsf{L} &\overset{\text{def}}{=} \mathsf{DSPACE}\left(\lg n\right) & \mathsf{NL} &\overset{\text{def}}{=} \mathsf{NSPACE}\left(\lg n\right) \\
\mathsf{PSPACE} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{DSPACE}\left(n^i\right) & \mathsf{EXPSPACE} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{DSPACE}\left(2^{n^i}\right)
\end{aligned}
$$

Table 2.1: Time and space complexity classes relevant for this thesis.

are defined as follows. Let $f : \mathbb{N} \to \mathbb{N}$,

$$
\begin{aligned}
\mathsf{DTIME}(f(n)) &\overset{\text{def}}{=} \{L \subseteq \Sigma^* : L = L(\mathcal{M}) \text{ for a deterministic } O(f(n))\text{-time TM } \mathcal{M}\} \\
\mathsf{NTIME}(f(n)) &\overset{\text{def}}{=} \{L \subseteq \Sigma^* : L = L(\mathcal{M}) \text{ for a non-deterministic } O(f(n))\text{-time TM } \mathcal{M}\} \\
\mathsf{DSPACE}(f(n)) &\overset{\text{def}}{=} \{L \subseteq \Sigma^* : L = L(\mathcal{M}) \text{ for a deterministic } O(f(n))\text{-space TM } \mathcal{M}\} \\
\mathsf{NSPACE}(f(n)) &\overset{\text{def}}{=} \{L \subseteq \Sigma^* : L = L(\mathcal{M}) \text{ for a non-deterministic } O(f(n))\text{-space TM } \mathcal{M}\}.
\end{aligned}
$$

Building upon those definitions, Table 2.1 defines the time and space complexity classes relevant for this thesis. We do not explicitly define $\mathsf{NPSPACE}$ and $\mathsf{NEXPSPACE}$ since by Savitch's theorem they are equivalent to PSPACE respectively $\mathsf{NEXPSPACE}$. For a given complexity class C, we denote by $\mathsf{coC} \overset{\text{def}}{=} \{\Sigma^* \setminus L : L \in \mathsf{C}\}$ the *complement class* of C, *e.g.*, coNP and $\mathsf{coNEXPTIME}$ are the complement classes of NP respectively $\mathsf{NEXPTIME}$. The complement class of Σ_1^0 is Π_1^0.

Non-deterministic Turing machines give rise to non-deterministic algorithms. Such algorithms can at any point during their execution branch into a finite number of child processes. A non-deterministic algorithm accepts an input if at any branching one child process accepts. When providing pseudo algorithms in thesis, we are going to use the additional primitive operation *existential move* in order to indicate the invocation of such a branching.

We close this section with the definition of reducibility between languages and languages that are complete for a complexity class. A function $f : \Sigma^* \to \Sigma^*$ is a *polynomial-time computable function* if there exists a deterministic polynomial-time

Turing machine $\mathcal{M}$ such that on any input $w \in \Sigma^*$, $\mathcal{M}$ accepts w and the content of the working tape of the accepting configuration is $\triangleright \cdot f(w)$. We say that a language $L \subseteq \Sigma^*$ is *polynomial-time reducible* to $L' \subseteq \Sigma^*$ if there exists a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for any $w \in \Sigma^*$, $w \in L$ if, and only if, $f(w) \in L'$. Given a language L and a complexity class C, we say L is *complete for* C *with respect to polynomial-time reductions* if, and only if, $L \in \mathsf{C}$ and every $L' \in \mathsf{C}$ is polynomial-time reducible to L.

2.4.1 Results from Structural Complexity Theory

In this section, we briefly introduce and recall results on alternative characterisations of PSPACE and $\mathsf{EXPSPACE}$ in terms of alternating Turing machines (ATM) and serialisability.

Alternation

A generalisation of non-deterministic Turing machines are *alternating Turing machines*, which were independently defined by Kozen and Chandra and Stockmeyer [24]. They give rise to *alternating algorithms*. Such algorithms can at any point during their execution branch into a finite number of child processes. There are two possible branching modes, *existential* and *universal* branching. Just as in the case of non-deterministic algorithms, in existential branching it is required that *one* of the child processes accepts, whereas in the universal mode the requirement is that *all* child processes accept. When providing pseudo-algorithms in this thesis, we are going to use the additional primitive operations *existential move* and *universal move* in order to indicate the invocation of branching of the respective type.

We do not give a formal definition of alternating Turing machines here as their precise definition does not have any direct relevance for this thesis. Informally speaking, an alternating Turing machine $\mathcal{M}$ is a non-deterministic Turing machine whose set of control locations is partitioned into a set $Q_\forall$ of *universal* and a set $Q_\exists$ of *existential* control locations. A configuration $C = (q, h_\mathsf{i}, h_\mathsf{w}, w_\mathsf{i}, w_\mathsf{w})$ is *accepting* if q is from the set of accepting configurations, or

- if $q \in Q_\exists$ then C is accepting if there is C' such that $C \to_{\mathcal{M}} C'$ and C' is accepting; or
- if $q \in Q_\forall$ then C is accepting if C' is accepting for all C' such that $C \to_{\mathcal{M}} C'$.

Given the initial configuration C of $\mathcal{M}$ for an input word $w \in \Sigma^*$, we say $\mathcal{M}$ accepts w if C is accepting. The complexity classes *alternating time* and *alternating polynomial time* are defined as follows:

$$\mathsf{ATIME}(f(n)) \overset{\text{def}}{=} \{L \subseteq \Sigma^* : L = L(\mathcal{M}) \text{ for an } O(f(n))\text{-time ATM } \mathcal{M}\}$$
$$\mathsf{AP} \overset{\text{def}}{=} \bigcup_{i \geq 0} \mathsf{ATIME}(n^i).$$

The following theorem shows that the languages in PSPACE coincide with languages in AP.

Theorem 2.4.1 ([**24**]) $\mathsf{AP} = \mathsf{PSPACE}$.

Serialisability

In this section, we provide an alternative characterisation of $\mathsf{EXPSPACE}$ in terms of serialisability. Let us begin with a generic notion of serialisability that is tailored to the needs of this thesis.

Definition 4 Let C be a complexity class and $R \subseteq \{0,1\}^*$. A language $L \subseteq \Sigma^*$ is *exponentially* C*-serialisable via* R if there exists a polynomial p and a language $U \in \mathsf{C}$ such that for all $w \in \Sigma^n$ and $m = \exp(p(n))$,

$$w \in L \Leftrightarrow \chi_U(w \cdot \mathrm{bin}_m(0)) \cdot \chi_U(w \cdot \mathrm{bin}_m(1)) \cdots \chi_U(w \cdot \mathrm{bin}_m(\exp^2(p(n)) - 1)) \in R.$$

$\Diamond$

Informally speaking, a language L is exponentially C-serialisable via R if deciding whether $w \in L$ can be reduced to a doubly exponential number of queries to a language $U \in \mathsf{C}$ with the requirement that the string of results of those queries, also known as the leaf language, is a word of some language R. This definition of serialisability is adopted from [50] and differs slightly from the standard notion that is used in [109, 58, 106].

The following theorem is due to Göller and provides a serialisability result for $\mathsf{EXPSPACE}$. The proof of the theorem relies on results from [50] and has only been published in an informal technical report accompanying [48]. In order to keep this thesis self-contained, we repeat the proof in the appendix.

Theorem 2.4.2 (Göller) *For every $L \in \mathsf{EXPSPACE}$ there is a regular language R such that L is exponentially L-serialisable via R.*

2.5 Models of Finite and Infinite-State Systems

This section introduces finite-state machines, counter automata and timed automata as mathematical models of finite and infinite state systems.

2.5.1 Finite-State Machines

Finite-state machines are a prominent mathematical model used for describing the behaviour of systems in the area of formal verification.

Definition 5 A *finite-state machine (FSM)* is a tuple $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda)$, where Q is a finite set of *control locations*, Λ is a finite set of labels, $q_0 \in Q$ is the *initial location*, $F \subseteq Q$ is the set of *final locations*, $\Delta \subseteq Q \times Q$ is the *transition relation*, and $\lambda : Q \to 2^\Lambda$ is the *locating labelling function.*

The definition of a finite-state machine is very similar to the finite automaton introduced in Section 2.3.1. The main difference is that we are not interested in a language accepted by a finite-state machine but rather in the words generated by traces of the labelled transition system that it induces. A finite-state machine $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda)$ induces the labelled transition system $T_\mathcal{A} = (S_\mathcal{A}, \to_\mathcal{A}, \Lambda, \lambda_\mathcal{A})$, where $S_\mathcal{A} \overset{\text{def}}{=} Q$, $\to_\mathcal{A} \overset{\text{def}}{=} \Delta$ and $\lambda_\mathcal{A} \overset{\text{def}}{=} \lambda$.

2.5.2 Counter Automata

Counter automata (CA) extend finite-state machines with a finite number of counters ranging over the naturals. At any transition, a counter automaton can increment a counter, decrement a counter provided that the resulting counter value is at least zero, or test if the value of a counter is zero.

Definition 6 Let $k \in \mathbb{N}_{>0}$ and $\mathsf{Op} \overset{\text{def}}{=} \{\mathsf{add}_i(z) : i \in [k], z \in \mathbb{Z}\} \cup \{\mathsf{zero}_i : i \in [k]\}$ be a set of *operations*. A *k-counter automaton* is a tuple $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda, \xi)$, where all components are the same as in Definition 5, except for $\xi : \Delta \to \mathsf{Op}$, which is an additional *transition labelling function.*

A k-counter automaton is called *zero-test free* if $\xi(q, q') \neq \mathsf{zero}_i$ for all $(q, q') \in \Delta$ and $i \in [k]$. We denote by $C(\mathcal{A}) \overset{\text{def}}{=} Q \times \mathbb{N}^k$ the set of all *configurations of $\mathcal{A}$*. The transition system generated by a k-counter automaton $\mathcal{A}$ is $T = (S_{\mathcal{A}}, \to_{\mathcal{A}}, \Lambda, \lambda_{\mathcal{A}})$, where $S_{\mathcal{A}} \overset{\text{def}}{=} C(\mathcal{A})$, $\lambda_{\mathcal{A}}(q, \vec{n}) \overset{\text{def}}{=} \lambda(q)$, and for $\vec{n} = (n_1, \ldots, n_k)$ and $\vec{n'} = (n'_1, \ldots, n'_k)$, $(q, \vec{n}) \to_{\mathcal{A}} (q', \vec{n}')$ if, and only if, $(q, q') \in \Delta$ and there exists an $i \in [k]$ such that

- $n'_j = n_j$ for all $j \in [k] \setminus \{i\}$; and
- $\xi(q, q') = \mathsf{add}_i(z)$ and $n'_i = n_i + z$; or
- $\xi(q, q') = \mathsf{zero}_i$ and $n_i = n'_i = 0$.

One of the earliest results about counter automata was obtained by Minsky who showed that reachability in counter automata is undecidable even when restricted to two counters only.

<u>Counter-Automata Reachability</u>

INPUT: A counter automaton $\mathcal{A}$ and $C, C' \in C(\mathcal{A})$.
QUESTION: Does $C \to^*_{\mathcal{A}} C'$?

Proposition 2.5.1 ([80]) *Reachability in k-counter automata is Σ^0_1-complete for $k \geq 2$.*

Note that for any counter automaton $\mathcal{A}$, it is easily seen that an arbitrary instance $(q, \vec{n}) \rightarrow^*_{\mathcal{A}} (q', \vec{n}')$ of a reachability problem can be reduced in polynomial time to a reachability instance $(q, \vec{0}) \rightarrow^*_{\mathcal{A}'} (q', \vec{0})$ in some counter automaton $\mathcal{A}'$. Reachability becomes decidable when we deal with zero-test free counter automata. This class of systems is also known as vector addition systems with states or Petri nets.

Proposition 2.5.2 ([78]) *Reachability in zero-test free counter automata is decidable.*

Another problem about counter automata that we consider in this thesis is the existence of a *Büchi run* or Büchi path. Given an infinite run $\varrho : (q_1, \vec{n}_1)(q_2, \vec{n}_2)\ldots$ of a counter automaton $\mathcal{A}$ in $T(\mathcal{A})$, let $\mathit{inf}(\varrho) \overset{\text{def}}{=} \{q \in Q : \text{for all } i \in \mathbb{N} \text{ such that } q_i = q$ there is a $j > i$ such that $q_j = q\}$. A run ϱ is a Büchi run in $T(\mathcal{A})$ if $\mathit{inf}(\varrho) \cap F \neq \emptyset$.

<u>Büchi Run of a Counter Automaton</u>

INPUT: A counter automaton $\mathcal{A}$ and $C \in C(\mathcal{A})$.
QUESTION: Does there exist a Büchi run ϱ in $T(\mathcal{A})$ such that $\varrho(1) = C$?

A more constrained class of counter automata are *bounded* counter automata in which an upper bound on the maximum value of each counter is imposed.

Definition 7 A *bounded k-counter automaton* is a tuple $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \vec{b}, \lambda, \xi)$, where all components are the same as in Definition 6 except for $\vec{b} \in \mathbb{N}^k$ which is a vector of *bounds.*

The set $C(\mathcal{A})$ of configurations of a bounded k-counter automaton with bounds $\vec{b} = (b_1, \ldots, b_k)$ is $C(\mathcal{A}) = Q \times [0, b_1] \times \ldots \times [0, b_k]$. Apart from that, the definition of the transition system induced by a bounded counter automaton is the same as for counter automata without bounds. Note that we can without loss of generality assume that no zero_i-labelled transitions occur in a bounded one-counter automaton, since any such transition can be replaced by two consecutive transitions that first add to and then subtract from the counter i the upper bound imposed on counter i.

A further class of counter automata that we consider are *parametric* counter automata. For example, in order to model underspecified systems, they generalise counter automata to allow for updates of the counter by some parametric value.

Definition 8 Let Y be a finite set of *parameters*. A *parametric k-counter automaton* is a tuple $\mathcal{A} = (Q, Y, \Lambda, q_0, F, \Delta, \lambda, \xi)$, where all components are the same as in Definition 6, but where the set of operations additionally allows for adding and subtracting parametric values, *i.e.*, $\mathsf{Op} \overset{\text{def}}{=} \{\mathsf{add}_i(z) : i \in [k], z \in \mathbb{Z}\} \cup \{\mathsf{add}_i(y), \mathsf{add}_i(-y) : i \in [k], y \in Y\} \cup \{\mathsf{zero}_i : i \in [k]\}$.

The set $C(\mathcal{A})$ of configurations of a parametric k-counter automaton $\mathcal{A}$ is defined in the same way as for k-counter automata, *i.e.*, $C(\mathcal{A}) \overset{\text{def}}{=} Q \times \mathbb{N}^k$. A parametric counter automaton represents an infinite family of counter automata, each of which is obtained from a *valuation* of the parameters, which is a function $\nu : Y \to \mathbb{N}$. Given a valuation ν and a parametric counter automaton $\mathcal{A} = (Q, Y, \Lambda, q_0, F, \Delta, \lambda, \xi)$, the counter automaton $\mathcal{A}^\nu$ is $\mathcal{A}^\nu \overset{\text{def}}{=} (Q, \Lambda, q_0, F, \Delta, \lambda, \xi')$, where for all $q, q' \in Q$, $\xi'(q, q') \overset{\text{def}}{=} \mathsf{add}_i(\circ\nu(y))$ if $\xi(q, q') = \mathsf{add}_i(\circ y)$ for $y \in Y$ and $\circ \in \{+, -\}$, and $\xi'(q, q') \overset{\text{def}}{=} \xi(q, q')$ otherwise. We call $\mathcal{A}^\nu$ the *counter automaton obtained from $\mathcal{A}$ under the valuation ν*. Reachability in parametric counter is now defined as follows:

<u>Parametric Counter-Automata Reachability</u>

INPUT: A parametric counter automaton $\mathcal{A}$ with parameters Y and $C, C' \in C(\mathcal{A})$.

QUESTION: Does there exist a valuation $\nu : Y \to \mathbb{N}$ such that $C \to^*_{\mathcal{A}^\nu} C'$?

For convenience, given a parametric counter automaton $\mathcal{A}$ and configurations $C, C' \in C(\mathcal{A})$, we write $C \to^*_{\mathcal{A}} C'$ if there exists a valuation ν such that $C \to^*_{\mathcal{A}^\nu} C'$.

Let us fix a parametric counter automaton $\mathcal{A} = (Q, Y, \Lambda, q_0, F, \Delta, \lambda, \xi)$. The *size* $|\mathcal{A}|$ of $\mathcal{A}$ is defined as follows: for the transition labels, $z \in \mathbb{Z}$ and $y \in Y$, we set $|\mathsf{add}_i(z)| \overset{\text{def}}{=} \lg |z|$, $|\mathsf{add}_i(\circ y)| = |\mathsf{zero}_i| \overset{\text{def}}{=} 1$, and finally

$$|\mathcal{A}| \overset{\text{def}}{=} |Q| + |\Delta| + \max\{|\xi(q, q')| : (q, q') \in Q\}.$$

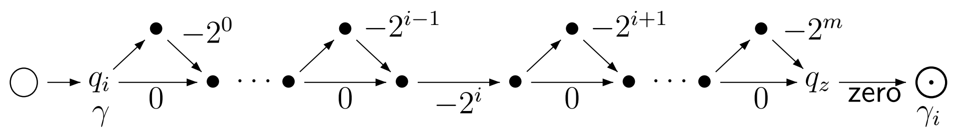

Figure 2.1: The one-counter automaton $\mathcal{A}_{m,i}$ used for testing a bit of a number $n \in [2^{m+1}-1]$.

Thus, our definition of size assumes binary encoding of numbers. If all counter updates are from the set $\{-1,0,1\}$, we call $\mathcal{A}$ a *unary* counter automaton. Otherwise, if we wish to emphasize binary encoding of numbers, we call $\mathcal{A}$ a *succinct* counter automaton.

In this thesis, we are going to establish a number of results on *succinct one-counter automata* and *parametric one-counter automata*. In this setting, it is always clear on which counter an operation is performed. For that reason we label, for example, an edge with "+5" or "+y" instead of "$\mathsf{add}_1(5)$" respectively "$\mathsf{add}_1(y)$", and with "zero" instead of "zero_1". Given a run $\varrho : (q_1,n_1)(q_2,n_2)\ldots$ in $T(\mathcal{A})$ of a one-counter automaton $\mathcal{A}$ such that no zero test occurs along ϱ, *i.e.*, $\xi(q_i,q_{i+1}) \neq \mathsf{zero}$ for all $i \in [|\varrho|-1]$, for technical convencience we denote for any $n \in \mathbb{N}$ by $\varrho + n$ the run $\varrho + n : (q_1, n_1+n)(q_2,n_2+n)\ldots$. Reachability and checking for the existence of a Büchi path for a unary one-counter automaton are known to be complete for NL.

Proposition 2.5.3 ([39]) *Reachability and checking for the existence of a Büchi run in unary one-counter automata is* NL*-complete.*

We are now going to consider an example of an instance of a reachability problem in a one-counter automaton and a parametric one-counter automaton. We are going to use the two examples in order to explain the way we graphically depict counter automata and how we represent and use *gadgets*. Let $m \in \mathbb{N}$, $i \in [0,m]$ and let us consider the one-counter automaton $\mathcal{A}_{m,i}$ presented in Figure 2.1. It consists of the control locations q_i, q_z and a number of further control locations, which are depicted as $\bullet$, $\bigcirc$ and $\odot$. A control location is labelled with the label(s) next to it, *e.g.*, q_i is labelled with $\{\gamma\}$, and if there is no label next to a control location it is implicitly

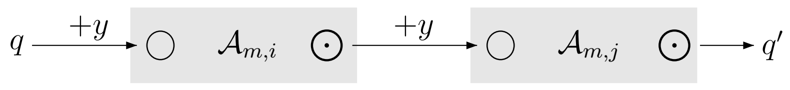

Figure 2.2: The parametric one-counter automaton $\mathcal{A}_{m,i,j}$, which illustrates the way gadgets are depicted in this thesis.

labelled with $\emptyset$. Transitions between control locations are depicted as arrows and are labelled with the operation that is performed along the transition. In Figure 2.1, there is a transition between q_z and $\odot$ which is labelled with zero. If there is no label along a transition, it is implicitly labelled with $+0$.

Let us now discuss the functionality of $\mathcal{A}_{m,i}$. Suppose we wish to analyse for which values of $n \in \mathbb{N}$, we have $(q_i, n) \to_{\mathcal{A}}^* (q_z, 0)$. Starting in (q_i, n), each triangle of $\mathcal{A}_{m,i}$ allows for non-deterministically subtracting 2^j *once* from the counter for each $j \neq i$, and when reaching the control location q_z it is required that 2^i has been subtracted from the counter. Thus, in order to be able to reach the configuration $(q_z, 0)$ starting in $(q_i, 0)$, n must not exceed $\sum_{i \in [0,m]} 2^m = 2^{m+1} - 1$ and the coefficient of 2^i in the binary representation of n must be 1. Hence the set of counter values n such that $(q, n) \to_{\mathcal{A}_{m,i}}^* (q_z, 0)$ can be characterised as follows:

$$\left\{n \in \mathbb{N} : (q_i, n) \to_{\mathcal{A}_{m,i}}^* (q_z, 0)\right\} = \left\{n \in [2^{m+1} - 1] : \mathrm{bit}_i(n) = 1\right\}.$$

Next, we discuss an example of a parametric one-counter automaton. Figure 2.2 depicts the parametric one-counter automaton $\mathcal{A}_{m,i,j}$, which uses the previously discussed one-counter automaton $\mathcal{A}_{m,i}$ respectively $\mathcal{A}_{m,j}$ as a gadget. The way this is graphically depicted is as follows: the grey-shaded boxes represent the whole $\mathcal{A}_{m,i}$ respectively $\mathcal{A}_{m,j}$ and the control locations $\bigcirc$ and $\odot$ are the control locations $\bigcirc$ and $\odot$ of $\mathcal{A}_{m,i}$ respectively $\mathcal{A}_{m,j}$. So, for example, $\mathcal{A}_{m,i,j}$ has a transition from q to $\bigcirc$ from $\mathcal{A}_{m,i}$ which adds the value of the parameter y to the counter. When dealing with gadgets, the control locations $\bigcirc$ and $\odot$ are always going to be used to mark the *entrance* respectively *exit* of a gadget, or initial and final location of the gadget. Since we can only reach $\odot$ in $\mathcal{A}_{m,i}$ and $\mathcal{A}_{m,j}$ when the value of the counter is zero, we can characterise the set of values of parameters for y such that $(q, 0) \to_{\mathcal{A}_{m,i,j}}^* (q', 0)$

as follows:

$$\Big\{n \in \mathbb{N} : \nu(y) = n, (q, 0) \to^*_{\mathcal{A}^{\nu}_{m,i,j}} (q', 0)\Big\} = \big\{n \in [2^{m+1} - 1] : \mathrm{bit}_i(n) = 1, \mathrm{bit}_j(n) = 1\big\}.$$

2.5.3 Timed Automata

Timed automata extend finite-state machines with a finite set of *clocks* ranging over the positive real numbers and were introduced by Alur and Dill [1]. While traces of paths in the transition system generated by a finite-state machine only allow for reasoning about the relative order of events, timed automata additionally incorporate timing information between them.

Let X be a finite set of *clock variables.* A *clock valuation* is a mapping $\vartheta : X \to \mathbb{R}_{\geq 0}$, and we denote by $CV(X) \stackrel{\text{def}}{=} \{\vartheta : \vartheta \text{ is a clock valuation}\}$ the *set of all clock valuations.* Given $r \in \mathbb{R}_{\geq 0}$, we denote by $\vartheta + r$ the clock valuation $\vartheta + r \stackrel{\text{def}}{=} x \mapsto \vartheta(x) + r$ for all $x \in X$. An *atomic clock constraint* is a term of the form $x \sim n$, where $x \in X$, $\sim \in \{<, \leq, =, \neq, \geq, >\}$ and $n \in \mathbb{N}$. A *clock constraint* ϕ is a finite conjunction of atomic clock constraints $\phi = x_1 \sim n_1 \wedge \ldots \wedge x_m \sim n_m$. The set of all clock constraints over clocks X is denoted by $CC(X)$. A clock valuation maps an atomic clock constraint $x \sim n$ to a Boolean value $\vartheta(x) \sim n$ and hence also a clock constraint ϕ to a Boolean value. We write $\vartheta \models \phi$ whenever ϑ makes ϕ true.

Definition 9 A *timed automaton* is a tuple $\mathcal{A} = (Q, X, \Lambda, q_0, F, \Delta, \lambda, \xi)$, where all components are the same as in Definition 5, except for X, which is a finite set of *clock variables* and $\xi : \Delta \to CC(X) \times 2^X$ which is the *transition labelling function* labelling each transition with a *guard.*

A guard is a tuple consisting of a clock constraint and a subset of the clock variables of $\mathcal{A}$ that are supposed to be reset when a transition is taken. We say that $\mathcal{A}$ is a k-clock timed automaton whenever $|X| = k$. Given a clock $x \in X$, the set of *x-constants C_x* is the set

$$C_x \stackrel{\text{def}}{=} \{n \in \mathbb{N} : \text{there are } q, q' \in Q \text{ s.t. } \xi(q, q') = (\phi, X') \text{ and } \phi \text{ has conjunct } x \sim n\} \cup \{0\}.$$

The set $C(\mathcal{A})$ of *configurations of* $\mathcal{A}$ is $C(\mathcal{A}) \stackrel{\text{def}}{=} Q \times CV(X)$ and consists of a control location and a clock valuation. A timed automaton induces a labelled transition system $T_{\mathcal{A}} = (S_{\mathcal{A}}, \to_{\mathcal{A}}, \Lambda, \lambda_{\mathcal{A}})$, where $S_{\mathcal{A}} \stackrel{\text{def}}{=} C(\mathcal{A})$, $\lambda_{\mathcal{A}}(q, \vartheta) = \lambda(q)$ and $(q, \vartheta) \to_{\mathcal{A}} (q', \vartheta')$ if one of the following conditions is satisfied:

(i) $q = q'$ and there exists $r \in \mathbb{R}_{\geq 0}$ such that $\vartheta' = \vartheta + r$; or

(ii) $(q, q') \in \Delta$, $\xi(q, q') = (\phi, X'), \vartheta \models \phi$ and ϑ' is such that $\vartheta'(x') = 0$ for every $x' \in X'$ and $\vartheta'(x) = \vartheta(x)$ for every $x \in X \setminus X'$.

Transitions of type (i) are called *delay transitions* and transitions of type (ii) *discrete transitions*. The *size of a timed automaton* is defined as

$$|\mathcal{A}| \stackrel{\text{def}}{=} |Q| + |\Delta| + \max\{\lg n : n \in C_x, x \in X\}.$$

As in the case of counter automata, we are interested in deciding reachability between configurations of a timed automaton.

<u>Timed-Automata Reachability</u>

INPUT: A timed automaton $\mathcal{A}$ with k clocks and $C, C' \in C(\mathcal{A}) \cap Q \times \mathbb{N}^k$.
QUESTION: Does $C \to_{\mathcal{A}}^* C'$?

Using a technique called *region abstraction*, Alur and Dill showed the following theorem.

Theorem 2.5.1 ([1]) *Reachability in timed automata is* PSPACE*-complete.*

This result was later refined by Courcoubetis and Yannakakis [37] who showed that PSPACE-hardness already holds if $\mathcal{A}$ comprises of three clocks. The cases with less than three clocks were left out in [37] and later discussed in [72], where it was shown that reachability in one-clock timed automata is NL-complete and NP-hard for two clocks. Closing the complexity gap for the case with two clocks is considered to be one of the biggest open problems in the theory of timed automata.

2.6 Integer Arithmetic

In this section, we are going to introduce and recall results on decidable and undecidable theories of number theory. The subsequent sections will in particular focus on the existential theories of Presburger arithmetic and Presburger arithmetic with divisibility, which we are going to use in Chapter 4 in order to show decidability and complexity results for reachability problems in one-counter and parametric one-counter automata.

The first-order theory of the natural numbers in the structure $\langle \mathbb{N}, <, +, \cdot, 0, 1\rangle$ was shown to be undecidable by Church [27]. Later, Matiyasevich proved Hilbert's tenth problem to be undecidable.

HILBERT'S TENTH PROBLEM (HTP)

INPUT: A polynomial $p : \mathbb{R}^n \to \mathbb{R}$ with integer coefficients.

QUESTION: Do there exist $a_1, \ldots, a_n \in \mathbb{Z}$ such that $p(a_1, \ldots, a_n) = 0$?

Theorem 2.6.1 ([77]) *Hilbert's tenth problem is Σ^0_1-complete.*

Since Hilbert's tenth problem can be expressed in the *existential* fragment of the structure $\langle \mathbb{N}, <, +, \cdot, 0, 1\rangle$, it follows that this theory is undecidable as well.

2.6.1 Presburger Arithmetic

Presburger arithmetic (PA) is the first-order theory of natural numbers in the structure $\langle \mathbb{N}, <, +, 0, 1\rangle$ and was shown to be decidable by Presburger [90] in 1929.

Let X be a countably infinite set of first-order variables. A *linear polynomial* over $\vec{x} = (x_1, \ldots, x_n) \in X^n$ is given by the syntax rule

$$p(\vec{x}) ::= \sum_{i \in [n]} a_i x_i + b,$$

where the a_i and b range over $\mathbb{Z}$ and the first-order variables from $\vec{x}$ range over $\mathbb{N}$. Formulae of Presburger arithmetic are defined by the following grammar where x

ranges over X:

$$\varphi ::= p(\vec{x}) < p(\vec{x}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\varphi.$$

We define the standard Boolean abbreviations $\varphi_1 \vee \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \to \varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2$ and $\varphi_1 \leftrightarrow \varphi_2 \stackrel{\text{def}}{=} \varphi_1 \to \varphi_2 \wedge \varphi_2 \to \varphi_1$. Moreover, we introduce the abbreviations $p_1(\vec{x}) \leq p_2(\vec{x}) \stackrel{\text{def}}{=} p_1(\vec{x}) < p_2(\vec{x})+1$ and $p_1(\vec{x}) = p_2(\vec{x}) \stackrel{\text{def}}{=} p_1(\vec{x}) \leq p_2(\vec{x}) \wedge p_2(\vec{x}) \leq p_1(\vec{x})$. Let $\vec{x} = (x_1, \ldots, x_n)$, for brevity we often write $\exists x_1 x_2 \ldots x_n.\varphi(\vec{x})$ instead of $\exists x_1.\exists x_2 \ldots \exists x_n.\varphi(\vec{x})$. Moreover, given a finite set $X = \{x_1, \ldots, x_n\}$ of first-order variables, we sometimes use a generalised existential quantifier and write $\exists_{x \in X}.\varphi(\vec{x})$ in order to abbreviate the formula $\exists x_1 \ldots x_n.\varphi(\vec{x})$.

The *size* $|\varphi|$ of a Presburger formula φ is defined by structural induction over φ. For a linear polynomial $p(\vec{x})$, its size $|p(\vec{x})|$ is the number of symbols needed to write it down, where we assume binary encoding of numbers. Now $|p_1(\vec{x_1}) < p_2(\vec{x_2})| \stackrel{\text{def}}{=} |p_1(\vec{x_1})| + |p_2(\vec{x_2})| + 1$, $|\varphi_1 \wedge \varphi_2| \stackrel{\text{def}}{=} |\varphi_1| + |\varphi_2| + 1$, $|\neg\varphi| \stackrel{\text{def}}{=} |\varphi| + 1$ and $|\exists x.\varphi| \stackrel{\text{def}}{=} |\varphi| + 1$. Note that binary encoding of numbers is not essential for complexity considerations since we can "simulate" binary encoding by introducing additional existentially quantified variables.

The set of *free variables* $\mathit{fv}(\varphi)$ of a formula φ is defined by structural induction on φ:

$$\begin{aligned}
\mathit{fv}(p(x_1, \ldots, x_n)) &\stackrel{\text{def}}{=} \{x_1, \ldots, x_n\} \\
\mathit{fv}(p_1(\vec{x_1}) < p_2(\vec{x_2})) &\stackrel{\text{def}}{=} \mathit{fv}(p_1(\vec{x_1})) \cup \mathit{fv}(p_2(\vec{x_2})) \\
\mathit{fv}(\varphi_1 \wedge \varphi_2) &\stackrel{\text{def}}{=} \mathit{fv}(\varphi_1) \cup \mathit{fv}(\varphi_2) \\
\mathit{fv}(\neg\varphi) &\stackrel{\text{def}}{=} \mathit{fv}(\varphi) \\
\mathit{fv}(\exists x.\varphi) &\stackrel{\text{def}}{=} \mathit{fv}(\varphi) \setminus \{x\}.
\end{aligned}$$

We write $\varphi(x_1, \ldots, x_n)$ to indicate that $\{x_1, \ldots, x_n\} \subseteq \mathit{fv}(\varphi)$. Without loss of generality, we assume that each first-order variable occurs at most once in the scope of an existential quantifier and that no first-order variable is both free and existentially quantified. Given $\varphi(x_1, \ldots, x_m)$ and $n_1, \ldots, n_m \in \mathbb{N}$, we write $\varphi[n_1/x_1, \ldots, n_m/x_m]$ for the formula obtained from replacing each x_i with n_i in φ. If $\mathit{fv}(\varphi) = \emptyset$, we write

$\langle \mathbb{N}, <, +, 0, 1\rangle \models \varphi$ if φ is a true statement in Presburger arithmetic, or just $\models \varphi$ if the structure we are working in is clear from the context.

In this thesis, we are going to show complexity results for reachability problems in one-counter automata via a translation into the *existential* or *quantifier-free fragment of Presburger arithmetic (QFPA).* This fragment restricts formulae to be of the form

$$\varphi = \exists x_1 \dots x_k . \psi(x_1, \dots, x_k)$$

and no quantifier is allowed to occur in ψ. Given a set $M \subseteq \mathbb{N}^k$, we say M *is QFPA-definable* if there exists a finite set R of QFPA formulae, each with free variables $x_1, \dots, x_k$, such that

$$M = \bigcup_{\varphi(x_1,\dots,x_k)\in R} \left\{(n_1, \dots, n_k) \in \mathbb{N}^k : \langle \mathbb{N}, <, +, 0, 1\rangle \models \varphi[n_1/x_1, \dots, n_k/x_k]\right\}.$$

Given a QFPA formula φ, checking whether $\models \varphi$, *i.e.*, if φ is satisfiable, is NP-complete.

Theorem 2.6.2 ([86]) *Satisfiability in quantifier-free Presburger arithmetic is* NP-*complete.*

We close this section with an example of a QFPA definable set. Recall the example of the one-counter automaton $\mathcal{A}_{m,i}$ in Section 2.5.2 which allows for testing whether $\mathrm{bit}_i(n) = 1$ for $n \in [0, 2^{m+1} - 1]$. The set $\{n \in \mathbb{N} : (q_i, n) \to^*_{\mathcal{A}_{m,i}} (q_z, 0)\}$ is definable via the QFPA formula

$$\varphi(n) \stackrel{\text{def}}{=} \exists x_1 \dots x_m . \bigwedge_{j\in[0,m]} (x_j = 0 \vee x_j = 1) \wedge n - \sum_{\substack{j\in[0,m]\\ j\neq i}} 2^j x_j - 2^i = 0.$$

2.6.2 Presburger Arithmetic with Divisibility

Presburger arithmetic with divisibility extends Presburger arithmetic with an additional predicate for divisibility, *i.e.*, it is the first-order theory of natural numbers in the structure $\langle \mathbb{N}, 0, 1, <, +, |\rangle$, where $a|b$ if there exists $k \in \mathbb{Z}$ such that $b = ka$ for $a, b \in \mathbb{N}$. Robinson [94] showed that the multiplication relation can be expressed in terms of the divisibility relation, which implies that this theory is undecidable due to the results discussed in the introduction of this section. However, the restriction to the *existential* or *quantifier-free fragment of Presburger arithmetic with divisibility (QFPAD)* makes the theory decidable, as independently shown by Lipshitz [74] and Bel'tyukov[8]. In fact, Lipshitz shows the decidability of formulae of the form

$$\varphi = \exists x_1 \dots x_k . \bigwedge_{i \in [n]} p_i(\vec{x}) | r_i(\vec{x}),$$

where the $p_i(\vec{x})$ and $r_i(\vec{x})$ are linear polynomials. However, this result implies the decidability of the full existential theory $\langle \mathbb{N}, 0, 1, <, +, | \rangle$ due to the following identities. Let $a, b \in \mathbb{Z}$, we can express the relations $=$ and $<$ in terms of conjunctions of divisibility relations:

$$\begin{aligned} a = b &\Leftrightarrow a|b \wedge b|a \wedge a+1|b+1 \wedge b+1|a+1 \\ a < b &\Leftrightarrow \exists z . a + z + 1 = b. \end{aligned}$$

Next, we consider the non-divisibility relation. First, if $|a| > |b|$ we have $a \nmid b$. Otherwise, we have

$$a \nmid b \Leftrightarrow \exists xyz . z|a \wedge z|b \wedge a|x \wedge b|y \wedge z = x - y \wedge 0 < z \wedge (z < a \vee z < -a).$$

Assuming $|a| < |b|$, informally speaking this identity states that $a \nmid b$ if a multiple z of the greatest common divisor of a and b is strictly less than $|a|$. In order to formally express this relation, we employ Bézout's identity, which states that for any pair $a, b \in \mathbb{Z}$, there are $x, y \in \mathbb{N}$ such that $\gcd(a, b) = ax - by$ and there do not exist $x', y' \in \mathbb{N}$ such that $n = ax' - by'$ for all $n \in [\gcd(a, b) - 1]$.

The definitions of size and free variables of a QFPAD formula φ are derived in a straight-forward manner from the corresponding QFPA definitions from the previous section, in particular $|p(\vec{x})|r(\vec{x})| \overset{\text{def}}{=} |p(\vec{x})| + |r(\vec{x})| + 1$. Likewise, given a set $M \subseteq \mathbb{N}^k$, we say *M is QFPAD-definable* if there exists a finite set R of QFPAD formulae, each with free variables $x_1, \dots, x_k$, such that

$$M = \bigcup_{\varphi(x_1, \dots, x_k) \in R} \left\{ (n_1, \dots, n_k) \in \mathbb{N}^k : \langle \mathbb{N}, 0, 1, <, +, | \rangle \models \varphi[n_1/x_1, \dots, n_k/x_k] \right\}.$$

Since QFPA is a notational fragment of QFPAD, satisfiability in this logic is NP-hard. In a follow-up paper [75], Lipshitz showed that satisfiability in QFPAD is NP-complete. In particular, he showed that NP-hardness already holds for a formula with five $\wedge$-connectives. It should however be noted that the paper [75] has been published in informal workshop proceedings. Although there is no good reason to doubt the results from [75], we wish to explicitly state at this point that the results therein have not been published in a peer-reviewed journal or in peer-reviewed conference proceedings.

Theorem 2.6.3 ([75]) *Satisfiability in quantifier-free Presburger arithmetic with divisibility is* NP*-complete and* NP*-hard already for a fixed number of Boolean connectives.*

Part I:
The Complexity of Model Checking Counter Automata

Chapter 3

Reachability Problems in Timed and Bounded Counter Automata

This chapter studies the relationship between reachability problems in classes of timed and counter automata. In short summary, we are going to show that an instance of a reachability problem in a timed automaton can be reduced to a reachability problem in a bounded two-counter automaton and *vice versa.* Figure 3.1 shows in more detail the precise relationships that we are going to establish. The arrows should be read as "reduces to". Thus, we are going to show that any reachability problem in a k-clock timed automaton reduces to a reachability problem in a bounded $2k + 2$-counter automaton, *etc.* A particularly special case that we are going to consider in the second part of this chapter are *two*-clock timed automata. We are going to show that reachability problems in this class are inter-reducible with reachability problems in bounded *one*-counter automata. This chapter requires the reader to have some level of familiarity with the *region abstraction technique* for timed automata, see *e.g.* [3] for an introduction. The general style of this chapter is a bit sketchy, the first part more, the second part less. We are going to put more emphasis on the presentation of the main ideas underlying the reductions than on the presentation of a bulk of otherwise unavoidable technical details. Nevertheless, we will get technical at the critical points. Knowledge of the region abstraction technique will allow the reader to develop further technical details if desired.

$$\begin{tikzcd}
 & \text{bounded } (2k+2)\text{-counter automata} \arrow[dr] & \\
k\text{-clock timed automata} \arrow[ur] & & \text{bounded two-counter automata} \arrow[dl] \\
 & \text{three-clock timed automata} \arrow[ul] &
\end{tikzcd}$$

Figure 3.1: Polynomial-time inter-reducibility between reachability problems in classes of timed and counter automata, where $k > 2$.

For technical convenience, in this chapter we are going to use a slightly modified version of bounded k-counter automata, which is equivalent with respect to reachability problems to bounded k-counter automata as defined in Definition 7. First, we allow for additional labelling of edges with $\mathsf{counter}_i \sim n, \sim \in \{\leq, <, =, >, \geq\}$ with the obvious semantics, where $i \in [k], n \in \mathbb{N}$ and . For example, an edge labeled with $\mathsf{counter}_i < n$ can only be taken if the counter value is strictly less than $n \in \mathbb{N}$. Let $\vec{b} = (b_1, \ldots, b_k)$ be the vector of bounds of a k-counter automaton, an edge labeled with $\mathsf{counter}_i < n$ can be simulated by two consecutive transitions, where the first adds $b_i - n + 1$ to the counter and the second subtracts $b_i - n + 1$. Moreover, in the second part of this chapter we are going to employ bounded one-counter automata whose counter updates and counter values are from $\mathbb{Z} \cup 0.5\mathbb{Z}$ and are bounded by an upper and a lower bound from $\mathbb{Z} \cup 0.5\mathbb{Z}$. It is easy to see that an instance of a reachability problem in such a bounded one-counter automaton can be reduced to standard one-counter automata by multiplying all numbers with two and adjusting the bounds accordingly.

3.1 From Bounded Counter Automata to Timed Automata and Back

We are now going to prove the reductions illustrated in Figure 3.1. All reductions and constructions in this section can be computed in polynomial time, and for readability we do not explicitly state this fact in the lemmas given in this section unless necessary. Polynomial-time computability is going to allow us to transfer complexity results regarding reachability problems between the classes of automata we consider in this section. We begin at the top of the figure and show that reachability in bounded k-counter automata, $k > 2$, reduces to reachability in bounded two-counter automata.

Let $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \vec{b}, \lambda, \xi)$ be a fixed bounded k-counter automaton for some $k \geq 2$. Our first result is that without loss of generality we can assume uniform bounds on the counters of $\mathcal{A}$.

Lemma 3.1.1 *Let $\mathcal{A}$ be a bounded k-counter automaton, let the bounds of $\mathcal{A}$ be $\vec{b} = (b_1, \ldots, b_n)$ and let $\hat{b} \geq \max\{b_1, \ldots, b_n\}$. There exists a bounded k-counter automaton $\mathcal{A}'$ such that the bounds of $\mathcal{A}'$ are $\vec{b'} = (\hat{b}, \ldots, \hat{b})$ and for all $(q, \vec{n}), (q', \vec{n}') \in C(\mathcal{A})$, $(q, \vec{n}) \to_{\mathcal{A}}^{*} (q', \vec{n}')$ if, and only if, $(q, \vec{n}) \to_{\mathcal{A}'}^{*} (q', \vec{n}')$.*

Proof. We sketch how to obtain the desired automaton $\mathcal{A}' \stackrel{\text{def}}{=} (Q', \Lambda, q_0, F, \Delta', \vec{b}', \lambda, \xi')$. For any edge $(q, q') \in \Delta$ such that $\xi(q, q') = \mathsf{add}_i(z), z \in \mathbb{Z}$, $\mathcal{A}'$ consists of an additional fresh control location (q, q'), and in order to obtain Δ' we remove any such transition (q, q') from Δ and add $(q, (q, q'))$ and $((q, q'), q')$ to Δ, *i.e.*, we split the edge (q, q') from $\mathcal{A}$. The labelling function ξ' is obtained by extending ξ such that $\xi'(q, (q, q')) \stackrel{\text{def}}{=} \mathsf{add}(z)$ and $\xi'((q, q'), q') \stackrel{\text{def}}{=} \mathsf{counter}_i \leq b_i$. □

We now show how we can reduce an instance of a reachability problem in a bounded k-counter automaton $\mathcal{A}$, $k > 2$, to a reachability problem in a bounded two-counter automaton $\mathcal{A}'$.

Lemma 3.1.2 *Let $\mathcal{A}$ be a bounded k-counter automaton with $k > 2$. There exists a bounded two-counter automaton $\mathcal{A}'$ such that for all $(q, \vec{n}), (q', \vec{n'}) \in C(\mathcal{A})$ there exist $\vec{m}, \vec{m'} \in \mathbb{N}^2$ such that $(q, \vec{n}) \to_{\mathcal{A}}^{*} (q', \vec{n'})$ if, and only if, $(q, \vec{m}) \to_{\mathcal{A}'}^{*} (q', \vec{m'})$.*

Proof. By the previous lemma we may assume with no loss of generality that $\mathcal{A}$ has a uniform bound $b = \exp(g) - 1$ for some $g \in \mathbb{N}$, hence $r \stackrel{\text{def}}{=} g - 1$ bits are sufficient to represent a counter value. The idea behind our reduction is to simulate the counters three to k of $\mathcal{A}$ in the upper bits of the second counter of $\mathcal{A}'$, and to use the upper bits of the first counter of $\mathcal{A}'$ as temporary storage.

The control locations of $\mathcal{A}'$ contain those of $\mathcal{A}$ as a subset, however the transitions of $\mathcal{A}$ are going to be replaced with gadgets in $\mathcal{A}'$. We set the bound of the counters of $\mathcal{A}'$ to $\exp(r(k-1)+1)-1$. In order to make our intuition about the relationship between configurations of $\mathcal{A}$ and $\mathcal{A}'$ formal, we define a mapping h as follows:

$$h : C(\mathcal{A}) \to C(\mathcal{A}') = (q, (n_1, \ldots, n_k)) \mapsto \left(q, \left(n_1, \sum_{i \in [2,k]} 2^{(i-2)r} n_i \right) \right).$$

Our aim is to construct $\mathcal{A}'$ such that $(q, \vec{n}) \to^*_{\mathcal{A}} (q', \vec{n'})$ if, and only if, $h(q, \vec{n}) \to^*_{\mathcal{A}'} h(q', \vec{n'})$. To this end, any transition (q, q') of $\mathcal{A}$ that adds a positive integer to the first counter, *i.e.*, is of the form $\mathsf{add}_1(n), n \in [0, b]$, gets replaced in $\mathcal{A}'$ by two consecutive transitions that first add n to the first counter of $\mathcal{A}'$ and then check that the value of this counter is less or equal to b. Any transition of $\mathcal{A}$ adding a negative number to the first counter is duplicated in $\mathcal{A}'$. Simulating the addition of integers to a counter different from the first counter requires some more efforts. Informally speaking, we have to make sure that we do not under- and overflow. Formally, any transition (q, q') labeled with $\mathsf{add}_i(z), i \geq 2, z \in \mathbb{Z}$, in $\mathcal{A}$ gets replaced in $\mathcal{A}'$ with a gadget that performs the following sequence of actions on the first and second counter of $\mathcal{A}'$ in this order:

(i) move the bits $(i-1)r+1$ up to $(k-1)r$ from the second to the first counter;

(ii) add $\exp((i-2)r)z$ to the second counter;

(iii) test that the value of the second counter is less than $\exp((i-1)r)+1$;

(iv) move the bits $(i-1)r$ up to $(k-1)r$ from the first to the second counter; and

(v) switch to control location q'.

Provided that all operations used in the gadget can be implemented, it is not difficult to verify that $(q, \vec{n}) \to_{\mathcal{A}} (q', \vec{n'})$ if, and only if, there is a path in $T(\mathcal{A}')$ traversing

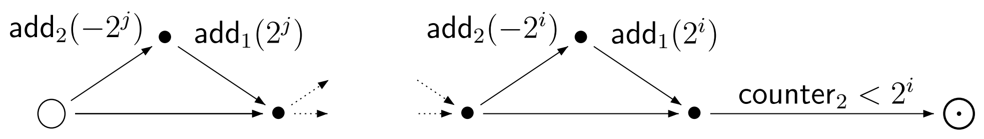

Figure 3.2: Gadget $\mathcal{A}_{mov}(i, j)$ used in the proof of Lemma 3.1.2 in order to move the bits from i up to j of the second to the first counter.

locations of the gadget starting in $h(q, \vec{n})$ and ending in $h(q', \vec{n'})$. It thus remains to discuss how the operations used in the gadget can be implemented.

Regarding (i), a gadget $\mathcal{A}_{mov}((i-1)r, (k-1)r)$ as sketched in Figure 3.2 can be used. The gadget non-deterministically subtracts relevant powers of two from the second counter and immediately adds them to the first counter. A test that the counter is less than $\exp((i-1)r)$ at the end ensures that all bits have been moved. The same gadget can be modified to move the same bits back from the first to the second counter. □

We now move one step forward in Figure 3.1 and show that reachability in bounded two-counter automata can be reduced to reachability in three-clock timed automata. Given a bounded two-counter automaton $\mathcal{A}$, the idea is to use the clocks x, y, z of a corresponding three-clock timed automaton $\mathcal{A}'$ in order to encode the value of the counters. By Lemma 3.1.1, we may assume that $\mathcal{A}$ has a uniform bound b. Our encoding is as follows: for any clock valuation ϑ, whenever $\vartheta(x) = b$ the value of the first counter of $\mathcal{A}$ is encoded in $\vartheta(x) - \vartheta(y)$ and $\vartheta(x) - \vartheta(z)$ encodes the second counter of $\mathcal{A}$. A similar encoding has also been used in [2] in order to show undecidability of reachability in parametric three-clock timed automata.

Lemma 3.1.3 *Let $\mathcal{A}$ be a bounded two-counter automaton and $(q, (n_1, n_2)), (q', (n_1', n_2')) \in C(\mathcal{A})$. There exists a three-clock timed automaton $\mathcal{A}'$ and ϑ, ϑ' such that $(q, (n_1, n_2)) \to_{\mathcal{A}}^{*} (q', (n_1', n_2'))$ if, and only if, $(q, \vartheta) \to_{\mathcal{A}'}^{*} (q', \vartheta')$.*

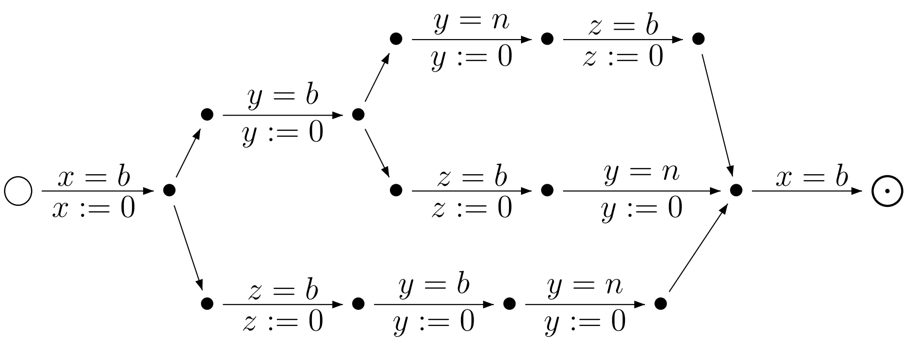

Figure 3.3: Gadget $\mathcal{A}_{inc,1}$ used to simulate incrementing the first counter by n in the proof of Lemma 3.1.3.

Proof. Without loss of generality, by Lemma 3.1.1 we can assume that $\mathcal{A}$ has a uniform bound b. The clock valuations ϑ, ϑ' required in the lemma are defined as $\vartheta(x) = \vartheta'(x) \overset{\text{def}}{=} b, \vartheta(y) \overset{\text{def}}{=} b - n_1, \vartheta(z) \overset{\text{def}}{=} b - n_2, \vartheta'(y) \overset{\text{def}}{=} b - n_1'$ and $\vartheta'(z) \overset{\text{def}}{=} b - n_2'$.

We are now going to sketch how $\mathcal{A}'$ can be obtained from $\mathcal{A}$. The timed automaton $\mathcal{A}'$ contains all control locations of $\mathcal{A}$. However, the transitions from $\mathcal{A}$ are going to be replaced by gadgets that manipulate the clocks in a way that simulates the action of the replaced transition. As an invariant, we are going to ensure that at any time $\mathcal{A}'$ reaches a control location that exists in $\mathcal{A}$, the value of the clock x is b. Suppose $(q, q') \in \Delta$ is a transition in $\mathcal{A}$ such that $\xi(q, q') = \mathsf{add}_1(n)$ for some $n \in \mathbb{N}$. In $\mathcal{A}'$, we are going to replace this transition by the gadget shown in Figure 3.3. In this figure, transitions are labeled with guards, *i.e.* with clock constraints such as $x = b$ and with clocks to be reset at a transition, *e.g.*, $x := 0$. Since we want to simulate that the first counter of $\mathcal{A}$ increases, we need to increase the difference between the value of the clock x and the value of the clock y. To this end, $\mathcal{A}_{inc,1}$ first resets the clock x. It then non-deterministically guesses the order of the simulated counter values: it branches upwards if the first counter is less or equal to the second counter and downwards otherwise. We are only going to discuss the first case here. $\mathcal{A}_{inc,1}$ waits until clock y has value b. It then aims at waiting for n time units. However, clock z could reach value b in the meantime. Thus, again, a non-deterministic choice is performed to handle the two cases. If z reaches b before y reaches n, the downward branch can be taken in $\mathcal{A}_{inc,1}$, which first resets z as it reaches clock value b and then y when it reaches clock value n. The converse case can be shown analogously. Finally, $\mathcal{A}_{inc,1}$ waits until clock x reaches clock value b in order to establish our agreed invariant when it reaches $\odot$, which is a control location present in $\mathcal{A}$.

A similar gadget can be constructed for the simulation of incrementing the second counter. Decrementing the counters can also be simulated in a similar fashion. For example, if we want to simulate decrementing the first counter by n, instead of waiting for y to reach clock value b, we wait for y to reach clock value $b - n$. This concludes our proof sketch. □

The only reduction from Figure 3.1 that remains to be shown is the reduction from reachability in k-clock timed automata to reachability in bounded $(2k + 2)$-counter automata. Let $\mathcal{A} = (Q, X, q_0, F, \Delta, \xi)$ be a timed automaton such that $X = \{x_1, \ldots, x_k\}$. Recall that a configuration of a timed automaton is a tuple consisting of a control state and clock valuation. In order to deal with the *a priori* infinite state space of a timed automaton, the region abstraction as a reachability preserving equivalence relation on the set of configurations of a timed automaton is defined in [1], which makes two configurations equivalent if

(a) their control location are the same;

(b) the integral part of the value of each clock with a value below the maximum constant appearing in $\mathcal{A}$ is the same;

(c) the relative order of the fractional parts of the value of the clocks is the same; and

(d) the clocks with fractional part 0 are the same.

We do not give further details of the region abstraction here and refer to [1, 3] for further information. Knowledge of this abstraction will however be helpful in understanding the reduction provided below as it is heavily inspired by it.

Given a k-clock timed automaton $\mathcal{A}$, we are now going to sketch how to construct a bounded $(2k + 2)$-counter automaton $\mathcal{A}'$ such that any reachability problem for

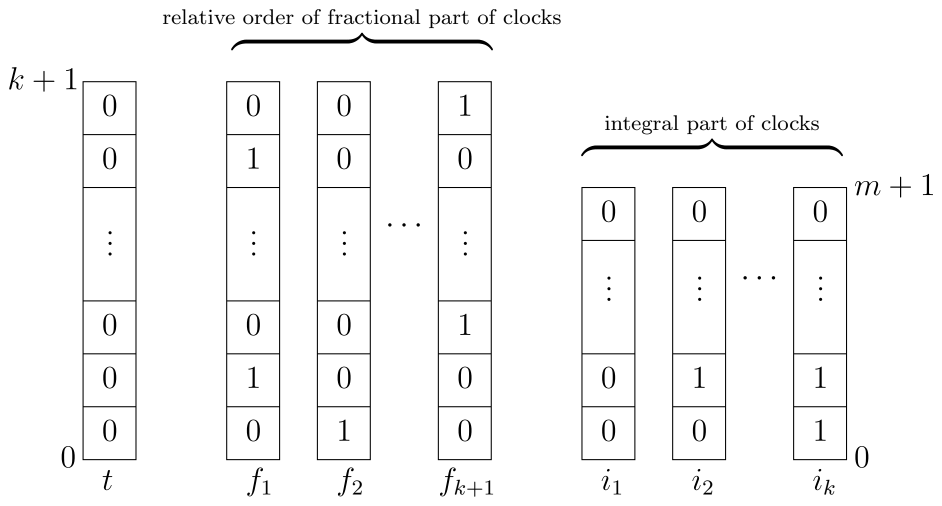

Figure 3.4: Encoding of the regions of a k-clock timed automaton into $2k+2$ counters.

$\mathcal{A}$ translates into an instance of a reachability problem in $\mathcal{A}'$. We aim for encoding (a)-(d) into configurations of $\mathcal{A}'$. Regarding (a), clearly the control locations of $\mathcal{A}$ can be included into $\mathcal{A}'$. However, any of (b)-(d) allows for an exponential number of possibilities in $|\mathcal{A}|$ and is therefore unsuitable to be encoded into control locations of $\mathcal{A}'$. Instead, we use the $2k+2$ counters for their encoding. Let $m \in \mathbb{N}$ be chosen such that m bits are sufficient to represent one plus the maximum integer constant appearing in $\mathcal{A}$. $\mathcal{A}'$ has bounded counters $f_1, \ldots, f_{k+1}$, $i_1, \ldots, i_k$ and t, where the maximum value for the counters $f_1, \ldots, f_{k+1}$ and t is $\exp(k+1) - 1$ and $\exp(m+1) - 1$ for the counters $i_1, \ldots, i_k$. The bit representation of the counters is illustrated in Figure 3.4, where the least significant bit of each counter is at the bottom and the most significant bit on top. The counter t is going to serve as temporary storage space. In order to represent a configuration (q, ϑ) of $\mathcal{A}$, $f_1, \ldots, f_{k+1}$ will be used to encode the order of the clocks with respect to their fractional parts induced by ϑ. The counter f_1 additionally encodes those clocks that have fractional part 0. Finally, the counters $i_1, \ldots, i_k$ will be used to store the integral part of the clocks induced by ϑ. We illustrate the encoding with the help of an example. Consider a clock valuation ϑ with $\vartheta(x_1) = 4.1$, $\vartheta(x_2) = 2.0$, $\vartheta(x_3) = 0.8$, $\vartheta(x_{k-1}) = 0.0$ and $\vartheta(x_k) = 3.8$. Let $l < l' \in [k]$, whenever the j-th bit of the counter f_l is set and the j'-th bit of the counter $f_{l'}$ is set, this is supposed to indicate that clock j has a value whose fractional part is strictly smaller than the fractional part of the value of clock j'. Combining our example with Figure 3.4, we see that the second bit of f_1 is set and the first bit of f_2, *i.e.*, the fractional part of clock x_2 is smaller than the fractional part of clock x_1 as expected. In addition, f_1 indicates which clocks have fractional part 0, which is why the second and the $(k-1)$-th bit of f_1 are set. Moreover, clock x_3 and x_k "reside" on the same counter f_{k+1} as their fractional part is equivalent in our example. The counters $i_1, \ldots, i_k$ are used to store the integral part of the clocks up to $\exp(m+1) - 1$. In our example, this means that the value of i_1 is 4, the value of i_2 is 2, *etc.* Elapsing of time can now be simulated as follows: first, the value of the counter f_{k+1} is moved to the counter t and the value of f_{k+1} is set to zero. Then, the value of the counter f_k is moved to the counter f_{k+1} until eventually we move the value of f_1 to f_2. We can then copy the value of t to f_1. All clocks that "resided" in f_{k+1} have now a fractional part zero and their integral part needs to be incremented by one. This can be simulated by incrementing the respective counter i_j, provided that it has not yet reached its maximum value. If the maximum value has already been reached, no action is performed. We defer the technical details to the next paragraph. In order to simulate $\mathcal{A}$, any control location of $\mathcal{A}$ is present in $\mathcal{A}'$ and has a loop which elapses time as described above. It remains to describe how to simulate a transition between two control locations of $\mathcal{A}$. To this end, checking the truth-value of the guard of the transition against the currently abstracted clock valuation and resetting of clocks needs to be simulated. Again, we illustrate the reduction with the help of an example. Suppose the guard is $(x_1 < 6 \wedge x_2 = 4, \{x_1\})$. The constraint $x_1 < 6$ can be checked in $\mathcal{A}'$ with an edge that is labeled with $\mathsf{counter}_{i_1} < 6$, checking $x_2 = 4$ can also be simulated with an edge $\mathsf{counter}_{i_2} = 4$, but we additionally need to check that the second bit of f_1 is set. Simulating a reset of x_1 is also relatively straight-forward: we non-deterministically choose the fractional class j of x_1, *i.e.*, the counter f_j whose first bit is set. We then set this bit to zero, *i.e.*, remove 2^0 from f_j, add 2^0 to the counter f_1 and set i_1 to zero. The latter can be implemented with the help of a loop that subtracts 1 from i_1 until a zero-test on i_1 is successful.

Let us now briefly discuss some of the technical details left out in the previous paragraph. In order to simulate elapsing of time, our reduction requires the possibility to move the contents between the counters of $\mathcal{A}'$. This task can easily be realised by a slight adoption of the gadget presented in Figure 3.2. Testing whether the a bit of a counter f_l is set can also be realised in a similar fashion. Using a gadget similar to the one in Figure 3.2, we first copy the value of the counter f_l to the counter t. Next, we use the gadget from Figure 2.1 in Chapter 2 on counter t in order to check if the bit is set.

In summary, in order to check $(q, \vartheta) \to_{\mathcal{A}}^{*} (q', \vartheta')$, we construct in polynomial time $\mathcal{A}'$, compute counter values $\vec{n}, \vec{n}' \in \mathbb{N}^{2k+2}$ that represent the abstraction of the clock valuations ϑ, ϑ' and check $(q, \vec{n}) \to_{\mathcal{A}'}^{*} (q', \vec{n}')$. The converse direction follows straight-forwardly by defining a bijection between configurations $(q, \vec{n})$ and the region abstraction of $\mathcal{A}$, we omit further details. We have thus proven the following lemma.

Lemma 3.1.4 *Let $\mathcal{A}$ be a k-clock timed automaton and $(q, \vartheta), (q', \vartheta') \in C(\mathcal{A})$. There exists a bounded $(2k+2)$-counter automaton $\mathcal{A}'$ and $\vec{n}, \vec{n}' \in \mathbb{N}^{2k+2}$ such that $(q, \vartheta) \to_{\mathcal{A}'}^{*} (q', \vartheta')$ if, and only if, $(q, \vec{n}) \to_{\mathcal{A}}^{*} (q', \vec{n}')$.*

The following theorem combines Lemmas 3.1.2, 3.1.3 and 3.1.4 and summarises all results obtained in this section.

Theorem 3.1.1 *Let $k > 2$. The following problems are polynomial-time reducible:*

- *reachability in k-clock timed automata to reachability in bounded $(2k+2)$-counter automata;*
- *reachability in bounded k-counter automata to reachability in bounded two-counter automata; and*
- *reachability in bounded two-counter automata to reachability in three-clock timed automata.*

As a byproduct, we obtain the complexity of reachability in bounded k-counter automata for $k \geq 2$.

Corollary 3.1.1 *Reachability in bounded k-counter automata is* PSPACE*-complete for $k \geq 2$.*

3.2 Two-Clock Timed Automata and Bounded One-Counter Automata

In the previous section, we dealt with timed automata with at least three clocks and bounded counter automata with at least two counters. In this section, we are going to consider the special case of two-clock timed automata and show that reachability for these automata is polynomial-time inter-reducible with reachability in bounded *one*-counter automata. This section is slightly more technical than the previous one since the reduction from reachability in two-clock timed automata to bounded one-counter automata requires some efforts to make sure that all constructions can be performed in polynomial time. Nevertheless, we will not sacrifice an understanding of the main ideas for providing all technical details.

The reduction from reachability in bounded one-counter automata to reachability in two-clock timed automata is a rather trivial adoption of the two-counter case presented in the previous section. Recall that in the reduction in Lemma 3.1.3 we encode the values of the counters into three clocks, the first counter is encoded as the difference between the clocks x and y, and the second counter as the difference between the clocks x and z. In the case of only one counter, two clocks are sufficient to store the value of the counter as the difference between the two clocks x and y. The rest follows from a straight-forward adaption of Lemma 3.1.3.

Lemma 3.2.1 *Reachability in bounded one-counter automata is polynomial-time reducible to reachability in two-clock timed automata.*

In the remainder of this section, we are thus going to concentrate on the reduction in the other direction. As a first step, let us provide a gadget that we will use in our reduction and that allows for adding numbers in an interval to the counter.

Lemma 3.2.2 *Let $a < b \in \mathbb{N}$. There exists a one-counter automaton $\mathcal{A}$ with control locations q, q' such that for all $n, n' \in \mathbb{N}$, $(q, n) \to_{\mathcal{A}}^{*} (q', n')$ if, and only if, $n' - n \in [a, b]$.*

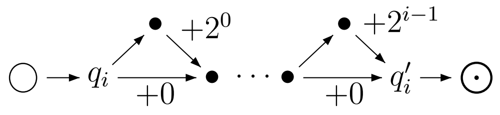

Figure 3.5: The one-counter automaton $\mathcal{A}_i$ used for adding a number in the interval $[2^i - 1]$ to the counter.

Proof. We first consider the case $a = 0$ from which we are then going to derive the general case. For any $m \in \mathbb{N}$, let $k(m) \overset{\text{def}}{=} \max\{i : m \geq (2^i - 1)\}$. We define a sequence $m_1 \geq m_2 \geq \ldots$ as follows:

$$m_1 \overset{\text{def}}{=} b$$
$$m_{i+1} \overset{\text{def}}{=} m_i - (2^{k(m_i)} - 1) \text{ for } i > 0.$$

Let $(k_i)_{i>0}$ be the sequence of the $k(m_i)$, we have $b = \sum_{i>0}(2^{k_i} - 1)$. Since $m_{i+1} \leq m_i/2$ for all $i > 0$, we have $k_{j+1} = 0$ for some $j \leq \lg b$ and hence $b = \sum_{i\in[j]}(2^{k_i} - 1)$. The one-counter automaton $\mathcal{A}$ consists of gadgets $\mathcal{A}_{k_i}, i \in [j]$ as shown in Figure 3.2.2 such that $\mathcal{A}_{k_i}$ connects to $\mathcal{A}_{k_{i+1}}$ for $i \in [j-1]$. For each $i \in \mathbb{N}$, on a run from $\bigcirc$ to $\odot$, $\mathcal{A}_i$ can non-deterministically add a number from the interval $[0, 2^i - 1]$ to the counter where we assume that $\mathcal{A}_0$ does not affect the counter at all. Let q be the initial location of $\mathcal{A}_{k_1}$ and q' the final location of $\mathcal{A}'_{k_j}$, it is easily verified that $(q, n) \rightarrow^*_{\mathcal{A}} (q', n')$ if, and only if, $n' - n \in [0, b]$.

In the general case where a and b take arbitrary values from $\mathbb{N}$, we construct a one-counter automaton $\mathcal{A}$ as above that allows for representing any number in the interval $[0, b - a]$ and add a new initial location that has a transition to the initial control location of $\mathcal{A}$ that adds a to the counter. □

Let $\mathcal{A} = (Q, X, q_0, F, \Delta, \xi)$ be a fixed two-clock timed automaton such that $X = \{x, y\}$. In the following, we will show how to construct in polynomial time a bounded one-counter automaton $\mathcal{A}' = (Q, \Lambda, q_0, F, \Delta, \vec{b}, \lambda, \xi)$ corresponding to $\mathcal{A}$. Here, we are going to use the modified bounded one-counter automaton described in the introduction to this chapter which allows for counter values and bounds from the set $\mathbb{Z} \cup 0.5\mathbb{Z}$.

The set of control locations Q' of $\mathcal{A}'$ is going to contain the control locations of $\mathcal{A}$ paired with abstractions of clock valuations. Thus, let us first define the abstractions that we are going to use. Let $C_x = \{x_1, \ldots, x_a\}$ be the ordered set of x-constants in $\mathcal{A}$, *i.e.*, $x_i < x_{i+1}$ for $i \in [a-1]$, and let $C_y = \{y_1, \ldots, y_b\}$ the ordered set of y-constants, where $x_1 = y_1 = 0$. We define the augmented sets C_x^∞ and C_y^∞ as $C_x^\infty \overset{\text{def}}{=} C_x \cup \{\infty\}$ respectively $C_y^\infty \overset{\text{def}}{=} C_y \cup \{\infty\}$, where x_{a+1} and y_{b+1} identify ∞ in C_x^∞ and C_y^∞, respectively. The set of *regions R of $\mathcal{A}$*, is defined as

$$R \overset{\text{def}}{=} \{(x_i, y_j, x_{i+b_x}, y_{j+b_y}) : x_i \in C_x, y_j \in C_y, b_x, b_y \in \{0,1\}\} \subseteq C_x \times C_y \times C_x^\infty \times C_y^\infty.$$

Note that $|R| = O(|\mathcal{A}|^2)$ and that R is computable in polynomial time. Subsequently, we will write r to identify a region from R. The abstraction of clock valuations provided by R can be obtained as follows. With each region $r \in R$, we associate a set of clock valuations $\vartheta(r)$, which is defined as

$$\begin{aligned}
\vartheta(x_i, y_j, x_i, y_j) &\overset{\text{def}}{=} \{\vartheta : \vartheta(x) = x_i, \vartheta(y) = y_j\} \\
\vartheta(x_i, y_j, x_{i+1}, y_j) &\overset{\text{def}}{=} \{\vartheta : x_i < \vartheta(x) < x_{i+1}, \vartheta(y) = y_j\} \\
\vartheta(x_i, y_j, x_i, y_{j+1}) &\overset{\text{def}}{=} \{\vartheta : \vartheta(x) = x_i, y_j < \vartheta(y) < y_{j+1}\} \\
\vartheta(x_i, y_j, x_{i+1}, y_{j+1}) &\overset{\text{def}}{=} \{\vartheta : x_i < \vartheta(x) < x_{i+1}, y_j < \vartheta(y) < y_{j+1}\}.
\end{aligned}$$

We observe that the set of regions R partitions the set of all clock valuations. Moreover, any two clock valuations of a region r cannot be distinguished by $\mathcal{A}$, *i.e.*, for any two $\vartheta, \vartheta' \in \vartheta(r)$ and any clock constraint ϕ occurring in labels of the transitions of $\mathcal{A}$, we have $\vartheta \models \phi$ if, and only if, $\vartheta' \models \phi$.

Figure 3.6 presents an example of the regions of a two-clock timed automaton $\mathcal{A}$ with $C_x = \{0, 1, 5\}$ and $C_y = \{0, 1, 3\}$. The stroked lines in the first quadrant indicate the regions of $\mathcal{A}$, *e.g.*, $(1, 1, 5, 3)$ and $(5, 3, \infty, \infty)$ are regions of $\mathcal{A}$.

A further abstraction that we are going to use builds upon the set of *clock differences D of $\mathcal{A}$*, which is defined as $D \overset{\text{def}}{=} \{c_x - c_y : c_x \in C_x, c_y \in C_y\}$. We write D as the ordered set $D = \{d_1, \ldots, d_c\}$. Our abstraction is the set of *clock difference zones Z of $\mathcal{A}$*, which is a set of symbolic intervals on $\mathbb{Z}$ defined as

$$Z \overset{\text{def}}{=} \{[d, d] : d \in D\} \cup \{(d_i, d_{i+1}) : d_i \in D, i \in [c-1]\} \cup \{[-\infty, d_1), (d_c, \infty]\}.$$

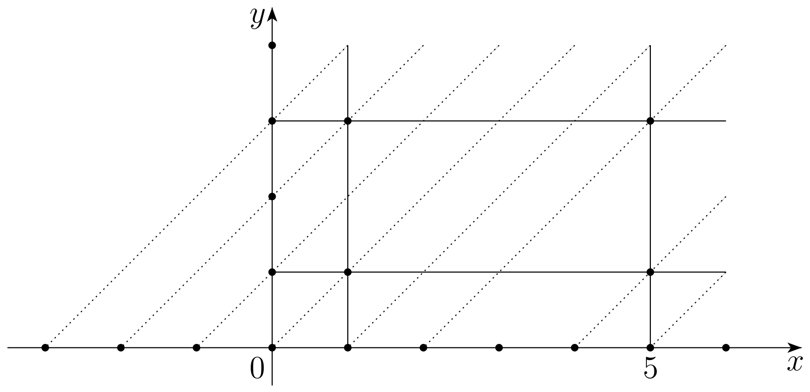

Figure 3.6: Example of the regions and the clock difference zones of a two-clock timed automaton with $C_x = \{0, 1, 5\}$ and $C_y = \{0, 1, 3\}$.

Here, we also have $|Z| = O(|\mathcal{A}|^2)$. We will subsequently write z to identify a clock difference zone from Z. With each z, we associate a set of clock valuations $\vartheta(z)$, which gives us an abstraction:

$$\vartheta(z) \overset{\text{def}}{=} \{\vartheta : \vartheta(x) - \vartheta(y) \in z\}.$$

The set of clock difference zones Z also partitions the set of all clock valuations. Figure 3.6 illustrates the partitioning of the clock valuations by clock difference regions where each dashed line and the space between them in the first quadrant is a partition.

We can now define a subset of the control locations of $\mathcal{A}'$. Our overall goal is to represent the set of configurations of $\mathcal{A}$ as a finite quotient encoded as configurations of $\mathcal{A}'$ and then discretely simulate transitions in $T(\mathcal{A})$ as transitions in $T(\mathcal{A}')$. In order to obtain the control locations Q' of $\mathcal{A}'$, we pair each $q \in Q$ with a region and a clock difference zone:

$$Q \times \{(r, z) \in R \times Z : \vartheta(r) \cap \vartheta(z) \neq \emptyset\} \subseteq Q'.$$

Each tuple $(q, (r, z))$ represents a set $\{(q, \vartheta) : \vartheta \in \vartheta(r) \cap \vartheta(z)\}$ of configurations of $\mathcal{A}$, and we can associate with each configuration (q, ϑ) a control location $(q, \vartheta)^\dagger$ of q' as follows:

$$(q, \vartheta)^\dagger \overset{\text{def}}{=} (q, (r, z)), \text{ where } r, z \text{ are uniquely chosen such that } \vartheta \in \vartheta(r) \cap \vartheta(z).$$

Given $r \in R$ and $z \in Z$ such that $\vartheta(r) \cap \vartheta(z) \neq \emptyset$, in order to discretely simulate time delay transitions of $\mathcal{A}$ in $T(\mathcal{A})$, we now define the *successor* $succ(r, z)$ *of* r *with respect to* z. Informally speaking, elapsing of time can be simulated by moving from region to region along the dashed lines in Figure 3.6. Let us first consider the case $z = [d, d]$ and suppose in the following that $i < a$ and $j < b$, *i.e.*, $x_{i+1} \neq \infty$ and $y_{j+1} \neq \infty$:

- case $r = (x_i, y_j, x'_i, y'_j)$, and $x'_i = x_i$ or $y'_j = y_j$: $succ(r, z) \overset{\text{def}}{=} (x_i, y_j, x_{i+1}, y_{j+1})$
- case $r = (x_i, y_j, x_{i+1}, y_{j+1}), x_{i+1} - y_{j+1} = d$: $succ(r, z) \overset{\text{def}}{=} (x_{i+1}, y_{j+1}, x_{i+1}, y_{j+1})$
- case $r = (x_i, y_j, x_{i+1}, y_{j+1}), x_{i+1} - y_{j+1} < d$: $succ(r, z) \overset{\text{def}}{=} (x_{i+1}, y_j, x_{i+1}, y_{j+1})$
- case $r = (x_i, y_j, x_{i+1}, y_{j+1}), x_{i+1} - y_{j+1} > d$: $succ(r, z) \overset{\text{def}}{=} (x_i, y_{j+1}, x_{i+1}, y_{j+1})$.

The definition of *succ* can straight-forwardly be extended for the cases in which $x_{i+1} = \infty$ or $y_{j+1} = \infty$. Now for the remaining case $z = (d_k, d_{k+1})$, we only sketch the definition of $succ(r, z)$. Again, suppose in the following that $i < a$ and $j < b$:

- case $r = (x_i, y_j, x_{i+1}, y_{j+1}), d_{k+1} \leq x_{i+1} - y_{j+1}$: $succ(r, z) \overset{\text{def}}{=} (x_i, y_{j+1}, x_{i+1}, y_{j+1})$
- case $r = (x_i, y_j, x_{i+1}, y_{j+1}), d_k \geq x_{i+1} - y_{j+1}$: $succ(r, z) \overset{\text{def}}{=} (x_{i+1}, y_j, x_{i+1}, y_{j+1})$

The remaining cases are defined analogously and it is not difficult to check that $succ(r, z)$ can be computed in polynomial time. In order to simulate time delay steps, $\mathcal{A}'$ contains transitions from each $(q, (r, z))$ to $(q, (succ(r, z), z))$ and to itself, which perform no action on the counter. Note that we can only simulate delay steps between regions but not within regions. Elapse of time inside regions only needs to be considered when resetting clocks and is going to be handled there. In order to handle clock resets, we are going to define a further abstraction that establishes a correspondence between clock valuations and counter values of $\mathcal{A}'$. For our construction, we use the modification discussed in the introduction to this chapter and allow the counter to take values from a bounded interval in $\mathbb{Z} \cup 0.5\mathbb{Z}$. More precisely, the counter of $\mathcal{A}'$ is bounded to take values from the set $V = \{d_1 - 0.5, d_1, \ldots, d_k, d_k + 0.5\}$. We use the counter to partition the set of clock valuations. For $n \in V$, we define

$$\vartheta(n) \overset{\text{def}}{=} \begin{cases} \{\vartheta : \vartheta(x) - \vartheta(y) = n\} & \text{if } n \in V \cap \mathbb{Z} \\ \{\vartheta : \vartheta(x) - \vartheta(y) \in (n - 0.5, n + 0.5)\} & \text{if } n \in V \setminus (\mathbb{Z} \cup \{d_1 - 0.5, d_k + 0.5\}) \\ \{\vartheta : \vartheta(x) - \vartheta(y) < d_1\} & \text{if } n = d_1 - 0.5 \\ \{\vartheta : \vartheta(x) - \vartheta(y) > d_k\} & \text{if } n = d_k + 0.5. \end{cases}$$

We will use this definition to map configurations of $\mathcal{A}$ to configurations of $\mathcal{A}'$. For any clock valuation ϑ, let ϑ^+ denote the unique $n \in V$ such that $\vartheta \in \vartheta(n)$. We define:

$$(q, \vartheta)^+ \overset{\text{def}}{=} ((q, \vartheta)^\dagger, \vartheta^+).$$

The partitioning of the clock valuations through the counter value is less coarse than through clock difference zones. It classifies clock valuations according to whether the difference between the clocks is a fixed integer, lies strictly in a unit interval between two consecutive fixed integers, or lies outside the "interesting" integers. While simulating $\mathcal{A}$ through $\mathcal{A}'$, we are going to ensure as an invariant that if we are in a configuration $((q, (r, z)), n)$ of $\mathcal{A}'$ then n is compatible with z, *i.e.*, $n \in z$. In fact, it is easy to construct a gadget that, informally speaking, non-deterministically finds out the clock difference zone the counter is currently in without destroying the counter value.

Let us informally justify with the help of an example why we need another abstraction of clock valuations through the counter value. Earlier, we have seen that regions and clock difference zones provide sufficient information to simulate the elapse of time. However, when it comes to simulating clock resets, the information they offer is insufficient. For example, consider Figure 3.6 and two clock valuations ϑ_1, ϑ_2 such that $r = (0, 1, 5, 1)$, $z = (2, 4)$, $n_1 = 2.5$, $n_2 = 3.5$, $\vartheta_1, \vartheta_2 \in \vartheta(r) \cap \vartheta(z)$, $\vartheta_1 \in \vartheta(n_1)$ and $\vartheta_2 \in \vartheta(n_2)$. If we let time elapse on ϑ_1 while staying in r, we cannot reach a point where, if we reset clock y, the value of the clock x is in the interval $(4, 5)$. Formally, for any $t_1 \in \mathbb{R}$ such that $\vartheta_1 + t_1 \in \vartheta(r)$, $((\vartheta_1 + t_1)[y \mapsto 0])(x) < 4$. This is however not the case for ϑ_2: there exists $t_2 \in \mathbb{R}$ such that $\vartheta_2 + t_2 \in \vartheta(r)$ and $((\vartheta_2 + t_2)[y \mapsto 0])(x) \in (4, 5)$. Even though ϑ_1 and ϑ_2 reside in the same region and clock difference zone, the clock difference zone we can reach through an elapse of time and by resetting the clock y is not fully determined only by the current region and zone and needs to take the abstraction through the counter value into account. Moreover, as we are going to see below, the change on the counter we need to perform in order to correctly simulate a clock reset only depends on the current region and zone, which is the crucial fact that allows for computing the transitions of $\mathcal{A}'$ in polynomial time.

We now give the technical details on how to simulate discrete transitions and clock resets. Throughout the remainder of this section, whenever we consider a configuration $((q,(r,z)),n)$ of $\mathcal{A}'$ that corresponds to some configuration (q,ϑ) of $\mathcal{A}$, it is helpful to think of ϑ to lie, if possible, at or, otherwise, infinitesimally close to the bottom left corner of $\vartheta(r) \cap \vartheta(n)$. In addition to the control locations mentioned above, Q' contains control locations that we are going to use to initiate the simulation of clock resets:

$$Q \times \{(r,z) \in R \times Z : \vartheta(r) \cap \vartheta(z) \neq \emptyset\} \times \{\mathit{reset}_x, \mathit{reset}_y, \mathit{reset}_{x,y}\} \subseteq Q'.$$

If $(q,q') \in \Delta$, $\xi(q,q') = (\phi, X')$ and $\vartheta \models \xi(q,q')$ for all $\vartheta \in \vartheta(r) \cap \vartheta(z)$ then, depending on which clocks are required to be reset by X', Δ' contains a transition from $(q,(r,z))$ to $(q',(r,z),\mathit{reset}_x)$, $(q',(r,z),\mathit{reset}_y)$ or $(q',(r,z),\mathit{reset}_{x,y})$, which perform no action on the counter. If no clock is required to be reset, *i.e.*, $X' = \emptyset$, then $(q,(r,z))$ directly connects to $(q',(r,z))$. Note that checking whether $\vartheta \models \phi$ for all $\vartheta \in \vartheta(r) \cap \vartheta(z)$ can be performed in polynomial time. The way we are going to simulating clock resets through $\mathcal{A}'$ requires a change of the counter value $\mathcal{A}'$. Thus, before we proceed with the simulation of clock resets, we are first going to relate configurations of $\mathcal{A}$ with configurations of $\mathcal{A}'$.

Let us first consider the simplest case in which we want to simulate a reset of both clocks x, y. This can be done by setting the counter to 0, changing r to $(0,0,0,0)$ and z to $[0,0]$. Thus, any $(q,(r,z),\mathit{reset}_{x,y})$ is connected to a gadget that non-deterministically increases and decreases the counter until the counter value is 0 and then connects to $(q,((0,0,0,0),[0,0]))$. If we only want to reset *one* clock,

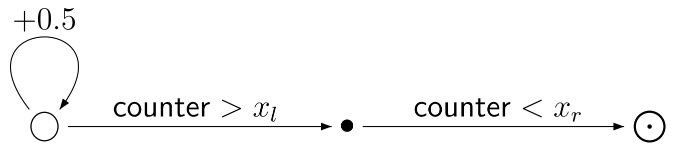

Figure 3.7: Gadget used to simulate a reset of clock y for the case when $r = (x_i, y_j, x_{i+1}, y_{j+1})$ and $z = [d, d]$.

things become slightly more complicated. As stated above, the range of the updated counter value then depends on the region and the clock difference zone. In the following, we are going to consider three representative cases that show how to simulate clock resets. The remaining cases follow a similar pattern.

First, suppose $r = (x_i, y_j, x_{i+1}, y_{j+1})$, $z = [d, d]$ and that we wish to reset the clock y of a clock valuation $\vartheta \in \vartheta(r) \cap \vartheta(z)$. Let us illustrate this case with the help of Figure 3.6, for example with $z = [0, 0]$ and $r = (1, 1, 5, 3)$. In this example, if we consider a clock valuation ϑ infinitesimally close to $(1, 1)$, if we let time elapse while staying inside r and then reset clock y, we obtain a new clock valuation ϑ' such that $\vartheta'(x) \in (1, 3)$ and hence $(q, \vartheta')^+ = ((q, (r', z')), n')$, where $r' = (1, 0, 5, 0)$, $z' \in \{(1, 2), [2, 2], (2, 3)\}$ and $n' \in [1.5, 2.5]$ such that z' and n' are compatible. Thus simulating a reset of clock y boils down to setting the counter to some value in the interval $[1.5, 2.5]$. This observation generalises to the following procedure: we pre-compute the left and right boundaries x_l, x_r on the x-axis of $\vartheta(r) \cap \vartheta(z)$, in our example 1 and 3 respectively, and connect $(q, (r, z), \mathit{reset}_y)$ to a gadget that non-deterministically repeatedly adds 0.5 to the counter, then performs a check that the counter value is strictly between x_l and x_r and finally non-deterministically performs a transition to the correct $(q, ((x_i, 0, x_{i+1}, 0), z'))$ for the new clock difference zone $z' = [d_k, d_k]$ or $z' = (d_k, d_{k+1})$ (recall that we can verify that we are in the correct clock difference zone.) The gadget that performs the counter update is illustrated in Figure 3.7. If we were to reset clock x, we pre-compute the lower and the upper boundaries y_l and y_u of $\vartheta(r) \cap \vartheta(z)$. We then non-deterministically subtract 0.5 from the counter, then ensure that the counter value is strictly between $-y_u$ and y_l and non-deterministically switch to the updated region and clock difference zone.

Next, we consider the case $r = (x_i, y_j, x_{i+1}, y_{j+1})$ and $z = (d_k, d_{k+1})$ where we wish to reset clock y. Again, we use Figure 3.6 to illustrate this case with the help of the region $r = (1, 1, 5, 3)$. Our first observation is that this case yields four different sub-cases. First, if $d = (-1, 0)$ then the boundaries of $\vartheta(r) \cap \vartheta(z)$ lie at the left and the top boundary of r. Second, if $d = (0, 2)$ then the boundaries of $\vartheta(r) \cap \vartheta(z)$ lie at the bottom and the top boundary of r. Third, if $d = (2, 4)$ then the boundaries of $\vartheta(r) \cap \vartheta(z)$ lie at the bottom and the right boundary of r. The fourth sub-case cannot be found in region $(1, 1, 5, 3)$ but in region $(0, 1, 1, 3)$, it is the case when the boundaries of $\vartheta(r) \cap \vartheta(z)$ lie at the left and the right boundary of r. Subsequently, we are going to consider the first and the second sub-case. The other sub-cases follow along similar lines.

Suppose $r = (x_i, y_j, x_{i+1}, y_{j+1})$, $z = (d_k, d_{k+1})$ and the boundaries of the intersection of $\vartheta(z)$ and $\vartheta(r)$ lie at (x_i, y_j, x_i, y_{j+1}) and $(x_i, y_{j+1}, x_{i+1}, y_{j+1})$, *e.g.*, $z = (-1, 0)$ in our example. Suppose $n \in V$ is the current counter value, since $\vartheta(y) < y_{j+1}$ for any $\vartheta \in \vartheta(r) \cap \vartheta(n)$, we have $\vartheta(x) < n + y_{i+1}$. This implies that when simulating a clock reset, the updated counter must not exceed $n + y_{i+1}$. On the other hand, the updated counter value must be above x_i. Thus, in this scenario, resetting clock y boils down to connecting $(q, (r, z), \mathit{reset}_y)$ to a gadget that adds y_{i+1} to the counter, non-deterministically subtracts 0.5 from the counter, checks whether the counter is strictly above x_i and then non-deterministically chooses the new z' that is compatible with the new counter value and switches to $(q, ((x_i, 0, x_{i+1}, 0), z'))$. If we were to reset clock x, we proceed analogously: we subtract x_i from the counter, non-deterministically subtract 0.5 and verify that the counter is strictly above $-y_{j+1}$.

The last case we consider is $r = (x_i, y_j, x_{i+1}, y_{j+1})$, $z = (d_k, d_{k+1})$ and $\vartheta(z)$ intersects with $\vartheta(r)$ at (x_i, y_j, x_{i+1}, y_j) and $(x_i, y_{j+1}, x_{i+1}, y_{j+1})$, *e.g.*, $z = (0, 2)$ in our example. Let us first consider resetting clock y. Similar to the previous case, we observe that for any $n \in z$ and $\vartheta \in \vartheta(r) \cap \vartheta(n)$, $\vartheta(x) < n + y_{j+1}$. Moreover, the lower bound for $\vartheta(x)$ is determined by y_j: $\vartheta(x) > n + y_j$. Thus, simulating a clock reset on clock y boils down to adding some number from the interval $[y_j + 0.5, y_{j-1} - 0.5]$ to the counter, which can be realised with the gadget from Lemma 3.2.2. In summary, in this case a clock reset on the clock y starting a control location $(q, (r, z), \mathit{reset}_y)$

can be simulated by connecting this control location to a gadget that adds a number from $[y_j + 0.5, y_{j-1} - 0.5]$ to the counter, then non-deterministically chooses the correct new clock difference zone z' and performs a transition to $(q, ((x_i, 0, x_{i+1}, 0), z'))$. If we were to reset clock x, we observe that the value of clock y always lies in the interval (y_i, y_{i+1}). Thus, starting in $(q, (r, z), \mathit{reset}_x)$, the reset can be simulated by connecting to a gadget that non-deterministically subtracts 0.5 from the counter and then verifies that the counter is strictly between $-y_{i+1}$ and $-y_i$.

All remaining cases have a symmetric case that we discussed before. It is not difficult to check that all constructions can be performed in polynomial time. The following lemma provides a summary of the properties of the reduction we described in this section and allows us to reduce reachability in two-clock timed automata to reachability in bounded one-counter automata.

Lemma 3.2.3 *Let $\mathcal{A}$ be a two-clock timed automaton, let $\mathcal{A}'$ be its corresponding bounded one-counter automaton and let $C = ((q, (r, z)), n), C' = ((q', (r', z')), n') \in C(\mathcal{A}')$. There exist ϑ, ϑ' such that $(q, \vartheta)^+ = C$, $(q', \vartheta')^+ = C'$ and $(q, \vartheta) \to^*_{\mathcal{A}} (q', \vartheta')$ if, and only if, $C \to^*_{\mathcal{A}'} C'$.*

In order to reduce an arbitrary instance $(q, \vartheta), (q', \vartheta')$ of a reachability problem in a two-clock timed automaton $\mathcal{A}$ to a reachability problem in a bounded one-counter automaton, we construct $\mathcal{A}'$ as described above, but use the sets $C_x \cup \{\vartheta(x), \vartheta'(x)\}$ and $C_y \cup \{\vartheta(y), \vartheta'(y)\}$ in order to construct the regions and clock difference zones of $\mathcal{A}'$. Applying the previous lemma, we obtain the main result of this section.

Theorem 3.2.1 *Reachability in two-clock timed automata polynomial-time inter-reducible with reachability in bounded one-counter automata.*

3.3 Discussion

This chapter discussed the relationship between reachability problems in timed automata and bounded one-counter automata. We have seen that for timed automata with at least three clocks, reachability reduces to reachability in bounded two-counter automata. Conversely, any instance of reachability in bounded two-counter automata reduces to reachability in three-clock timed automata. All reductions can be performed in polynomial time. We have additionally considered the special case of reachability in two-clock timed automata and shown that this problem is inter-reducible with reachability in bounded one-counter automata. This dichotomy can even extend to one-clock timed automata: Laroussinie *et al.* show [72] that reachability in this class is reducible in polynomial time to a reachability problem in a directed graph. The latter can be viewed as an instance of a reachability problem in a counter automaton with no counters.

It has been observed in the early days of timed automata that there is a relationship between timed and counter automata. The classical undecidability proof of the universality problem for timed automata by Alur and Dill [1] proceeds via a reduction from reachability in two-counter automata. Moreover, Alur, Henzinger and Vardi use the same problem to show undecidability of reachability in parametric timed-automata with at least three parameterised clocks [2]. However, to the best of the author's knowledge, no direct correspondence between reachability problems in timed and counter automata has been known, and this gap has been closed in this chapter.

While timed automata are a useful tool for modeling systems that require explicit timing information, we believe that algorithmic properties of verification problems can more easily be analysed for (bounded) counter automata since their definition is much simpler than the definition of timed automata. In particular with respect to settling the complexity of reachability in two-clock timed automata, the reduction to reachability in bounded one-counter automata provided in Section 3.2 considerably simplifies this problem. At this point, it is fair to mention that both problems have independently been investigated [72] and [16] without observing that they are essentially the same.

An interesting aspect for future work would be to find a similar correspondence for reachability problems in parametric timed automata and parametric bounded counter automata. By parametric bounded counter automata we mean bounded counter automata whose counters can be updated by some parametric value and where the vector of bounds can also contain parameters. In [2], it has been mentioned that the reachability problem for a rather non-standard class of parametric one-counter automata reduces to reachability in parametric two-clock timed automata, but not *vice versa*. Lifting the correspondence between two-clock timed automata and bounded one-counter automata to the parametric case by adopting the construction presented in Section 3.2 of this chapter should be possible if there is a way to construct a parameterised variant of the gadget constructed in Lemma 3.2.2. This correspondence might be helpful in order to show that reachability in parametric two-clock timed automata is decidable, which is an open problem. For the case of parametric timed automata with more than two clocks, an adoption of the reduction from Section 3.1 should be a rather straight-forward task.

Chapter 4

Reachability in Counter Automata

This chapter studies the computational complexity of reachability in classes of counter automata. In particular, we consider reachability for one-counter automata, parametric counter automata and bounded one-counter automata.

The first section shows that reachability in one-counter automata is NP-complete, where obtaining the upper bound is the more difficult part. Given a one-counter automaton $\mathcal{A}$ and control locations q, q', we are going to show that the reachability set

$$\big\{(n, n') \in \mathbb{N}^2 : (q, n) \to_{\mathcal{A}}^{*} (q', n')\big\}$$

is definable via a set $R_{\mathcal{A}}(q, q')$ of QFPA formulae, where each formula in $R_{\mathcal{A}}(q, q')$ is of size polynomial in the size of $\mathcal{A}$ and can be *guessed* in non-deterministic polynomial time, which yields membership of reachability in NP. The construction of $R_{\mathcal{A}}(q, q')$ is quite involved. For that reason the section is broken into four parts.

The next section discusses reachability in parametric counter automata. Our first result is that reachability in parametric counter automata is undecidable in the presence of four counters even if we disallow zero tests. On the positive side, we subsequently proceed by showing that reachability in parametric one-counter automata is decidable and in fact NP-complete. This result heavily depends on the results and techniques obtained in the first section and cannot be understood without having read the first section. Similar to the first section, membership in NP is shown by showing that given a parametric one-counter automaton $\mathcal{A}$ with parameters $y_1, \ldots, y_k$ and control locations q, q', the reachability set

$$\Big\{(n_1, \ldots, n_k, n, n') \in \mathbb{N}^{k+2} : (q, n) \to^*_{\mathcal{A}^\nu} (q', n') \text{ for a valuation } \nu \text{ such that } \nu(y_i) \overset{\text{def}}{=} n_i\Big\}$$

can be defined via a set $R_{\mathcal{A}}(q, q')$ of QFPAD formulae, where the size of each formula in $R_{\mathcal{A}}(q, q')$ is polynomial in the size of $\mathcal{A}$ and can be guessed in non-deterministic polynomial time.

The third section then considers reachability in bounded counter automata. In the previous chapter, we have already shown that reachability in bounded counter automata with at least two counters is PSPACE-complete. Consequently, this section focuses on bounded one-counter automata. The precise computational complexity of reachability in this class remains an open problem of this thesis. We are going to discuss a very simple class of bounded one-counter automata for which we are unable to determine the precise complexity of reachability and provide an approach that might be helpful one day for settling the complexity of this problem.

We close this chapter with a discussion of the results obtained, how they fit into the existing literature and have been used there, and discuss some directions for future work.

4.1 One-Counter Automata

In this section we are going to show that reachability in one-counter automata is NP-complete. As one-counter automata can be viewed as pushdown automata acting on a singleton alphabet and reachability in pushdown automata is decidable, reachability in one-counter automata is decidable.

For complexity considerations, it is essential that we assume numbers to be encoded in binary. Probably due to their close relationship to pushdown automata, research has mainly focused on one-counter automata with numbers encoded in unary. Reachability in this class of counter automata is NL-complete, see *e.g.* [71]. A paper by Rosier and Yen [95] is one of the first papers to consider one-counter automata with numbers encoded in binary, this time from the perspective of vector addition systems with states. Amongst other things, their paper is concerned with the *boundedness problem* for one-counter automata, which is the question to decide for a given one-counter automaton $\mathcal{A}$ and a configuration (q, n) whether the reachability set

$$\{(q', n') \in Q \times \mathbb{N} : (q, n) \to_{\mathcal{A}}^{*} (q', n')\}$$

is infinite. The authors leave the precise computational complexity of this problem open, but claim that it is NP-complete:

> “We surmise, but are unable to show, that the aforementioned problem is solvable in NP. [...] The best we can do, at this time, then, is to deduce that the problem is doable in PSPACE.”

This section gives a positive answer to their claim. Our result that reachability in one-counter automata is NP-complete yields, as a corollary, that deciding boundedness for one-counter automata is NP-complete.

Lafourcade *et al.* show in [71], Lemma 42, that given a one-counter automaton $\mathcal{A}$, (q, n) and (q', n'), if $(q, n) \to_{\mathcal{A}}^{*} (q', n')$ then there exists a path $\varrho : (q, n) \to_{\mathcal{A}}^{*} (q', n')$ in $T(\mathcal{A})$ such that no counter value along ϱ exceeds a value polynomial in m, n and n', where m is the maximum increment occurring in $\mathcal{A}$. Since numbers are encoded in binary, this implies that a path witnessing reachability has length at most exponential in $|\mathcal{A}|$ and the binary representation of n and n'. The following example shows that witnessing paths of exponential length can actually not be avoided. Let $m, n \in \mathbb{N}$ and consider the following one-counter automaton $\mathcal{A}$:

$$\begin{tikzcd} q \arrow[loop above, "+1"] \arrow[r, "-2^m"] & \bullet \arrow[r, "\mathsf{zero}"] & q' \arrow[loop above, "+1"] \arrow[r, "-2^n"] & \bullet \arrow[r, "\mathsf{zero}"] & q'' \end{tikzcd}$$

Suppose we wish to decide $(q, 0) \to_{\mathcal{A}}^{*} (q'', 0)$, a path witnessing reachability has to traverse the self-loop at q 2^m times and the self-loop at q' 2^n times, which makes the length of such a path exponential in $|\mathcal{A}|$. In order to show that reachability in one-counter automata is NP-complete, this rules out the most natural approach of using a witnessing path as a certificate.

On a technical level, in order to show NP membership of reachability in one-counter automata, it is more convenient for our purposes to work with weighted graphs instead of one-counter automata. In order to decide reachability in one-counter automata, we can restrict ourselves to zero-test-free one-counter automata since each transition testing for zero only needs to be traversed at most once, and the order in which those transitions are traversed can be guessed in NP. In the example above, deciding $(q, 0) \to_{\mathcal{A}}^{*} (q'', 0)$ reduces to checking $(q, 0) \to_{\mathcal{A}}^{*} (q', 0)$ and $(q', 0) \to_{\mathcal{A}}^{*} (q'', 0)$. Zero-test-free one-counter automata can then be viewed as weighted graphs.

One of the main techniques to provide a polynomial-size certificate witnessing reachability is to succinctly describe paths in weighted graphs as *path flows*. A path flow assigns a natural number to each transition indicating how often a transition is traversed on a path witnessing reachability. In the example above, viewing $\mathcal{A}$ as two weighted graphs by deleting the two zero-labelled edges, a path flow asserts that the self-loop at q is traversed 2^m times, the self-loop at q' 2^n times and all other transitions are traversed once. For one-counter automata whose control structure is more complex, a path flow in its corresponding weighted graph alone is not sufficient to prove reachability. In this section, we will carefully analyse paths and corresponding path flows and provide sufficient and necessary conditions on path flows that prove the existence of a path witnessing reachability in the original transition system of the one-counter automaton under consideration. We are going to show that checking the existence of path flows and all necessary conditions can be defined via sets of QFPA formulae of polynomial size and can each formula can be guessed in NP. Since satisfiability in QFPA is NP-complete, this is eventually going to show that reachability in one-counter automata is NP-complete.

The structure of this section is as follows. The first section establishes NP-hardness of reachability in one-counter automata. Although this fact has already been shown in [95], we show that reachability is already NP-hard for a one-counter automaton with very little structure. Hardness is shown via a generalisation of the SUBSET-

Sum problem. The second section then introduces weighted graphs and some related concepts such as paths and weights of paths. We also show that some properties of weighted graphs can be expressed in QFPA. The third section introduces path flows as a way to succinctly represent paths in weighted graphs. We are also going to show some technical lemmas about decomposition of path flows that we are going to use in the remainder of this chapter. The fourth section then combines the concepts of weighted graphs and path flows, applies them to one-counter automata and shows that reachability relations can be defined via sets of polynomial-size QFPA formulae. This result is then extended in order to show that checking for the existence of Büchi paths and determining boundedness is also NP-complete.

As we have to make sure that all constructions can be performed in (non-deterministic) polynomial time, this section is the lengthiest and most technical section of this thesis. Like in the previous chapter, for the sake of lucidity, when giving constructions in this section we often do not explicitly state that they can be performed in polynomial time.

4.1.1 The NP Lower Bound

In this section we are going to show that reachability for one-counter automata is NP-hard. This result follows already as a corollary from similar results in the literature, *e.g.* from [95, 72]. However, we would like to emphasize in this section that reachability is NP-hard even for one-counter automata with an underlying control graph with very little structure.

The basis for our proof of the lower bound is the well-known NP-complete SubsetSum problem, see *e.g.* [100], which is defined as follows:

SubsetSum

INPUT: A set $S = \{n_1, \ldots, n_m\} \subseteq \mathbb{N}$ and a *target* $t \in \mathbb{N}$.
QUESTION: Does there exist $S' \subseteq S$ such that $\sum_{n \in S'} n = t$?

For the NP-hardness of SubsetSum, binary encoding of numbers is essential. For our purposes, we introduce a slight generalisation of the SubsetSum problem which

S'	P	P_m	$\cdots$	P_2	P_1	Z	D_k	$\ldots$	D_2	D_1
n'_1	1	0	$\cdots$	0	1	0	$\mathrm{dig}_k(n_1)$	$\cdots$	$\mathrm{dig}_2(n_1)$	$\mathrm{dig}_1(n_1)$
p_1	1	0	$\cdots$	0	1	0	0	$\cdots$	0	0
n'_2	1	0	$\cdots$	1	0	0	$\mathrm{dig}_k(n_2)$	$\cdots$	$\mathrm{dig}_2(n_2)$	$\mathrm{dig}_1(n_2)$
p_2	1	0	$\cdots$	1	0	0	0	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
n'_m	1	1	$\cdots$	0	0	0	$\mathrm{dig}_k(n_m)$	$\cdots$	$\mathrm{dig}_2(n_m)$	$\mathrm{dig}_1(n_m)$
p_m	1	1	$\cdots$	0	0	0	0	$\cdots$	0	0
t'	m	1	$\cdots$	1	1	0	$\mathrm{dig}_k(t)$	$\cdots$	$\mathrm{dig}_2(t)$	$\mathrm{dig}_1(t)$

Table 4.1: The input to MultiSubsetSum obtained from an input to SubsetSum. All numbers are encoded in base $m + 1$.

we call MultiSubsetSum.

MultiSubsetSum

INPUT: A set $S = \{n_1, \ldots, n_m\} \subseteq \mathbb{N}$ and a *target* $t \in \mathbb{N}$.
QUESTION: Does there exist $f : S \to \mathbb{N}$ such that $\sum_{n \in S} nf(n) = t$?

MultiSubsetSum differs from SubsetSum in that an element of the set S can contribute more than once to the final target value. However, one can construct instances which prevent any element of S from being picked more than once, which allows to show NP-hardness of MultiSubsetSum via a reduction from SubsetSum.

Proposition 4.1.1 MultiSubsetSum *is* NP*-complete.*

Proof. Membership in NP is trivial. Let S, t be an input to MultiSubsetSum. The description of a witnessing function is of polynomial size in the size of S and t, since $f(n) \leq t$ for each $n \in S$.

In order to show NP-hardness, we reduce from SubsetSum. Let $S = \{n_1, \ldots, n_m\}, t$ be an input to SubsetSum. Without loss of generality, we may assume that $n_i < t$ for all $i \in [m]$. From S, t, we construct an input $S' = \{n'_i, p_i\}_{i \in [m]}, t'$ to MultiSubsetSum such that S', t' has a solution if, and only if, S, t has a solution. The construction is given in Table 4.1. Apart from the last row, every row in the table is an n'_i respectively p_i, and starting from the second column, every column corresponds to one digit of n_i respectively p_i. The last row is the target t' that we construct. Every n'_i, p_i and t is encoded in *base* $m+1$, and $k-1$ is the power of the highest non-zero power of $m+1$ in the base $m+1$ representation of t, *i.e.*, $t = \sum_{i \in [k]} d_{i-1}(m+1)^{i-1}$ for some $d \in \{0, \dots, m\}$. Each digit of a constructed number is identified by the identifier at the top row. For example, D_1 is the digit of the lowest power of a constructed number, so D_1 of t' is $\mathrm{dig}_1(t)$. All n'_i correspond in any D_j to their n_i counterpart, and the p_i are all zero there. Each n'_i has two more non-zero digits, P_i and P, which are also the only non-zero digits of each p_i. The D_j digits of t' also correspond to those of t, the digit P is m, Z is zero and all other digits are one. It can easily be seen that S', t' can be computed in polynomial time from S, t.

Let $S'' \subseteq S$ be a solution to the instance of SubsetSum S, t. For each $n_i \in S''$, we set $f(n'_i) \overset{\text{def}}{=} 1$ and $f(p_i) \overset{\text{def}}{=} 0$. For each $n_i \notin S''$, we set $f(n'_i) \overset{\text{def}}{=} 0$ and $f(p_i) \overset{\text{def}}{=} 1$. It can easily be checked that f is a solution to S', t', since by assumption the digits $D_1, \dots, D_k$ sum up correctly, $f(n'_i) + f(p_i) = 1$ for all $i \in [m]$, and hence the digits P_j and P also sum up correctly to 1 respectively m.

Conversely, assume that f is a solution to S', t'. We show that $f(n'_i) + f(p_i) = 1$ for all $i \in [m]$. It is clear that this implies that $S'' \overset{\text{def}}{=} \{n_i : f(n'_i) = 1\}$ is a solution to S, t. First, we see that $\sum_{s \in S'} f(s) \leq m$, since otherwise we have a carry over at digit P. Let $r \overset{\text{def}}{=} (m+1)^k$ and recall that $\bar{n}_r$ denotes the residue class of $n \in \mathbb{N}$ modulo r, we have $\sum_{s \in S'} f(s)\bar{s}_r < (m+1)^{k+1}$, since

$$\begin{aligned}
\sum_{s \in S'} f(s)\bar{s}_r &\leq m(m \sum_{i \in [k]} (m+1)^{i-1}) \\
&< (m+1)(m \sum_{i \in [k]} (m+1)^{i-1}) \\
&= m \sum_{i \in [k]} (m+1)^i \\
&< (m+1)^{k+1}
\end{aligned}$$

This implies that every digit greater than k of $\sum_{s \in S'} f(s)\bar{s}_r$ is zero. Hence $f(n'_1) +$

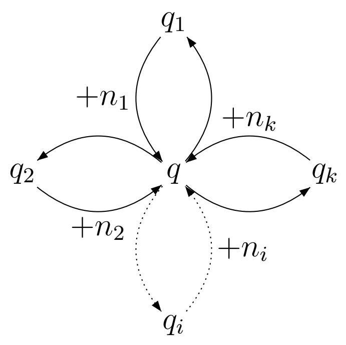

Figure 4.1: The OCA $\mathcal{A}$ constructed in the reduction from an input $S = \{n_1, \ldots, n_k\}, t$ to MultiSubsetSum such that S, t has a solution if, and only if, (q, t) is reachable from $(q, 0)$.

$f(p_1') \geq 1$, since the digit P_1 of t' is one. Moreover, since $f(n_1') + f(p_1) \leq m$, we must have $f(n_1') + f(p_1) = 1$. Iterating this argument for the remaining digits P_i and n_i' and p_i, it follows that $f(n_i') + f(p_i) = 1$ for all $i \in [m]$. □

Proposition 4.1.2 *Reachability in one-counter automata is* NP*-hard.*

Proof. We reduce from MultiSubsetSum. Let $S = \{n_1, \ldots, n_k\}, t$ be an input to MultiSubsetSum. Figure 4.1 shows the one-counter automaton $\mathcal{A}$ constructed from S and t. For each $n_i \in S$, $\mathcal{A}$ has a transition from q to q_i and back that adds n_i to the counter. Suppose there is $\varrho : (q, 0) \to_{\mathcal{A}}^{*} (q, t)$. Counting the number of times a transition from q to q_i occurs in ϱ gives rise to the required witnessing function f that shows that S, t has a solution. Conversely, in the same way any such function allows for constructing a path witnessing $\varrho : (q, 0) \to_{\mathcal{A}}^{*} (q, t)$. □

The graph underlying the one-counter automaton from Figure 4.1 has a very simple structure. In fact, if we would allow for *multiple* transitions between control locations, this means that reachability is already NP-hard even for a one-counter automaton consisting of only one control location. Note that by employing a similar reduction from SubsetSum used in [72] to show NP-hardness of reachability in two-clock timed automata, reachability for one-counter automata is also NP-hard for one-counter automata whose underlying graph is acyclic.

4.1.2 Weighted Graphs

In order to show an NP-upper bound of the reachability problem, we subsequently rely on some basic concepts from graph theory which we are going to introduce below. Let us start with giving a formal definition of weighted graphs.

Definition 10 A *weighted graph* is a tuple $G = (V, E, \mu)$, where V is a finite set of *vertices*, $E \subseteq V \times V$ is a finite set of *edges*, and $\mu : E \to \mathbb{Z}$ is a *weight function.* $\Diamond$

Subsequently, we call weighted graphs just graphs. The *size* $|G|$ *of a graph* G is defined as

$$|G| \overset{\text{def}}{=} |V| + |E| + \max\{\lg |z| : \mu(e) = z \text{ for some } e \in E\}.$$

We call $G' = (V', E', \mu')$ a *subgraph* of G if $V' \subseteq V$, $E' \subseteq E$ and $\mu'(e') = \mu(e')$ for all $e' \in E'$. For a given graph $G = (V, E, \mu)$, any subset $E' \subseteq E$ induces a subgraph $G/E' \overset{\text{def}}{=} (V', E', \mu')$, where

- $V' \overset{\text{def}}{=} \{v \in V : \text{there is } v' \in V \text{ such that } (v, v') \in E \text{ or } (v', v) \in E\}$
- $\mu'(e) \overset{\text{def}}{=} \mu(e)$ for all $e \in E'$

The *skew transpose* G^{op} of a graph G is the graph $G^{\mathsf{op}} \overset{\text{def}}{=} (V, E^{\mathsf{op}}, \mu^{\mathsf{op}})$, which is obtained from flipping every edge and the sign of every edge of G. Formally,

- $E^{\mathsf{op}} \overset{\text{def}}{=} \{(v, w) : (w, v) \in E\}$; and
- $\mu^{\mathsf{op}}(v, w) \overset{\text{def}}{=} -\mu^{\mathsf{op}}(w, v)$ for all $(v, w) \in E^{\mathsf{op}}$.

Clearly, both a subgraph and the skew transpose of a graph can be computed in polynomial time. For the remainder of this section, let us fix a weighted graph $G = (V, E, \mu)$.

Similar to transition systems, an *s-t path* π in G of length n is a function $\pi : [n+1] \to V$ or, alternatively, a non-empty sequence of vertices $\pi = v_1 v_2 \dots v_{n+1}$ such that $v_1 = s$, $v_{n+1} = t$ and $(v_i, v_{i+1}) \in E$ for all $i \in [n]$. We write $|\pi|$ to denote the length of π. For $i \le j \in [|\pi|+1]$, $\pi(i,j)$ denotes the path $\pi(i,j) \overset{\text{def}}{=} \pi(i)\pi(i+1)\dots\pi(j)$. We say t is reachable from s and write $s \to_G^* t$ if there is an s-t path in G. Given an s-u path $\pi = v_1 \dots v_m$ and a u-t path $\pi' = v'_1 \dots v'_n$, the s-t path $\pi \cdot \pi'$ is defined as $\pi \cdot \pi' \overset{\text{def}}{=} v_1 \dots v_m v'_2 \dots v'_n$.

We say that a graph is *connected* if for any $v, v' \in V$, v' is reachable from v via some path. The *set of edges traversed by a path* π is defined to be

$$\mathit{edges}(\pi) \overset{\text{def}}{=} \{e \in E : \text{there is } i \in [|\pi|] \text{ such that } (\pi(i), \pi(i+1)) \in E\}.$$

Given a path $\pi = v_1 \dots v_{n-1} v_n$ in a graph G, we denote by $\pi^{\mathsf{op}} \overset{\text{def}}{=} v_n v_{n-1} \dots v_1$ the *skew transpose of* π, which is a path in G^{op}.

A path π is a *simple path* if any vertex of a graph occurs at most once along π. If the first and the last vertex of a path ℓ is the same vertex v, we call it a *v-cycle* or a *v-loop*. Note that in particular a zero-length path consisting of only one vertex is a cycle. For any v-cycle π and $n \in \mathbb{N}$, the v-cycle ℓ^n is defined by induction on n as $\ell^0 \overset{\text{def}}{=} v$ and $\ell^{n+1} \overset{\text{def}}{=} \ell^n \cdot \ell$. If v is the only vertex occurring twice along a v-cycle ℓ, we call ℓ a *simple cycle*. A graph G *is a loop* if it is connected and there is exactly one simple v-cycle between any vertex v of G. We call G a *simple s-t path* if G is not connected and a cycle when (t, s) is added to the set of edges.

One central property of a path π in a weighted graph is its *weight*. It is the sum over all weights of the edges occurring along π. Formally, for $|\pi| > 0$

$$\mathit{weight}(G, \pi) \overset{\text{def}}{=} \sum_{i \in [|\pi|]} \mu(\pi(i), \pi(i+1)).$$

If $|\pi| = 0$ then $\mathit{weight}(G, \pi) \overset{\text{def}}{=} 0$. The minimum accumulated weight of all prefixes of a path π is called the *drop of* π, which is formally defined as

$$\mathit{drop}(G, \pi) \overset{\text{def}}{=} \min\{\mathit{weight}(G, \pi(1, i)) : i \in [|\pi| + 1]\}.$$

If G is clear from the context, we simply write $\mathit{weight}(\pi)$ and $\mathit{drop}(\pi)$ to denote the weight respectively drop of π.

The next lemma shows that both the weight and the drop of a path in a graph can be expressed in terms of an open formula in QFPA, which are polynomial time computable.

Lemma 4.1.1 *Given a path π in a graph G, there exist QFPA formulae $\varphi_w(G,\pi)(c,c')$ and $\varphi_d(G,\pi)(d,d')$ such that*

- *$\varphi_w(G,\pi)[n/c, n'/c']$ is satisfiable if, and only if, weight$(\pi) = n' - n$, and*
- *$\varphi_d(G,\pi)[n/c, n'/c']$ is satisfiable if, and only if, drop$(\pi) = n' - n$.*

Moreover, $|\varphi_w| = O(|\pi|)$ and $|\varphi_w| = O(|\pi|^2)$.

Proof. It is easily checked that the following two QFPA formulae have the desired properties:

$$\begin{aligned}
\varphi_w(G,\pi)(c,c') &\overset{\text{def}}{=} \sum_{i\in[|\pi|]} \mu(\pi(i),\pi(i+1)) = c' - c\\
\varphi_d(G,\pi)(c,c') &\overset{\text{def}}{=} \bigvee_{i\in[|\pi|+1]} \varphi_w(G,\pi(1,i))(c,c') \wedge\\
&\qquad \wedge \bigwedge_{i\in[|\pi|+1]} (\exists d,d'.\varphi_w(G,\pi(1,i))(d,d') \wedge d' - d \geq c' - c)\,.
\end{aligned}$$

□

If *weight*$(\ell) > 0$ for a given cycle ℓ then we call ℓ a *positive cycle*, and if *weight*$(\ell) < 0$ then ℓ is called a *negative cycle*. Likewise, if *weight*$(\ell) \geq 0$ then ℓ is called a *weakly positive cycle*, and if *weight*$(\ell) \leq 0$ then we call ℓ a *weakly negative cycle*.

In the remainder of this section, we establish some facts about the properties of paths, their weight and drop that we are going to use in the remainder of this and the next chapter. The following technical lemma contains a collection of statements about the relationship between paths and their weight and drop. These statements will be helpful when relating paths in weighted graphs to runs in one-counter automata.

Lemma 4.1.2 *Given a graph G, paths π, π', a weakly positive cycle ℓ and $n \in \mathbb{N}$, the following statements hold:*

(i) $\mathit{drop}(\pi \cdot \pi') \geq -n$ *if, and only if,* $\mathit{drop}(\pi) \geq -n$ *and* $\mathit{weight}(\pi) + \mathit{drop}(\pi') \geq -n$*;*

(ii) $\mathit{drop}(\ell) \geq -n$ *if, and only if,* $\mathit{drop}(\ell^k) \geq -n$ *for all* $k > 0$*; and*

(iii) $\mathit{drop}(\pi^{\mathsf{op}}) = \mathit{drop}(\pi) - \mathit{weight}(\pi)$*.*

Proof. (i) The statement follows from the following equalities:

$$\begin{aligned}
&\mathit{drop}(\pi \cdot \pi') \geq -n \\
\iff& \min\{\mathit{weight}(\pi \cdot \pi'(1,i)) : i \in [|\pi \cdot \pi'|]\} \geq -n \\
\iff& \min\{\mathit{weight}(\pi(1,i)), \mathit{weight}(\pi) + \mathit{weight}(\pi'(1,j)) : i \in [|\pi|+1], j \in [|\pi'|+1]\} \geq -n \\
\iff& \min\{\mathit{drop}(\pi), \mathit{weight}(\pi) + \mathit{drop}(\pi')\} \geq -n \\
\iff& \mathit{drop}(\pi) \geq -n \text{ and } \mathit{weight}(\pi) + \mathit{drop}(\pi') \geq -n.
\end{aligned}$$

(ii) We show the statement by induction on k. For the induction step, we have that

$$\begin{aligned}
&\mathit{drop}(\ell^{k+1}) \geq -n \\
\iff& \mathit{drop}(\ell^k \cdot \ell) \geq -n \\
\overset{\textit{by (i)}}{\iff}& \mathit{drop}(\ell^k) \geq -n \text{ and } \mathit{weight}(\ell^k) + \mathit{drop}(\ell) \geq -n \\
\iff& \mathit{drop}(\ell) \geq -n.
\end{aligned}$$

(iii) First, it can easily be seen that for all $i \in [|\pi|+1]$,

$$\mathit{weight}(\pi(1,i)) = \mathit{weight}(\pi) + \mathit{weight}(\pi^{\mathsf{op}}(1, |\pi|+2-i)).$$

Hence we conclude that

$$\begin{aligned}
\mathit{drop}(\pi) &= \min\{\mathit{weight}(\pi(1,i) : i \in [|\pi|+1]\} \\
&= \min\{\mathit{weight}(\pi) + \mathit{weight}(\pi^{\mathsf{op}}(1, |\pi|+2-i)) : i \in [|\pi|+1]\} \\
&= \mathit{weight}(\pi) + \min\{\mathit{weight}(\pi^{\mathsf{op}}(1, |\pi|+i)) : i \in [|\pi^{\mathsf{op}}|+1]\} \\
&= \mathit{weight}(\pi) + \mathit{drop}(\pi^{\mathsf{op}}).
\end{aligned}$$

□

We devote the remainder of this section to some results on positive cycles in graphs. Positive cycles can be used to construct paths whose weight exceeds any natural number, which is a fact we are going to exploit when proving the NP upper bound of reachability in one-counter automata. We first give some criteria that prove the existence of positive cycles in a graph. Next, we consider how to algorithmically determine if there is a positive cycle in a graph.

Given a vertex v, as seen in the introduction to this section, in the worst case a positive v-cycle can be of exponential length. However, if we are only interested in witnessing the *existence* of a positive cycle at v, it is sufficient to give a not-necessarily positive cycle of linear length as a certificate.

Definition 11 Let ℓ be a v-cycle and $n \in \mathbb{N}$. We call ℓ a *positive v-cycle template with respect to n* if ℓ decomposes into $\ell = \pi_1 \cdot \pi_2 \cdot \pi_3$ such that π_2 is a positive w-cycle, $drop(\pi_1 \cdot \pi_2) \geq -n$ and $0 \leq |\pi_1|, |\pi_2|, |\pi_3| \leq |G|$. $\diamondsuit$

We are now going to show that a positive cycle template proves the existence of a positive cycle in a graph.

Lemma 4.1.3 *Let $v \in V$ and $n \in \mathbb{N}$. There exists a positive v-cycle template ℓ with respect to n if, and only if, there exists a positive v-cycle ℓ' such that $drop(\ell') \geq -n$.*

Proof. ("$\Rightarrow$") Since $weight(\pi_2) > 0$ and $drop(\pi_1 \cdot \pi_2) \geq -n$ we can always find $k > 0$ such that for $\ell' \overset{\text{def}}{=} \pi_1 \cdot {\pi_2}^k \cdot \pi_3$ we have $weight(\ell') > 0$. Lemma 4.1.2(i) implies that $drop(\ell') \geq -n$.

("$\Leftarrow$") Let ℓ' be a positive v-cycle of length m with $drop(\ell) \geq -n$. Without loss of generality we may assume that ℓ' does not contain any negative cycles. Let i, j be chosen minimal such that $\pi_2 = \ell(i, j)$ is a simple positive w-cycle for some vertex w. Let $\pi_1 = \ell(1, i)$ and π_3 be obtained from $\ell(j, m+1)$ by removing all cycles. We have that $\ell \overset{\text{def}}{=} \pi_1 \cdot \pi_2 \cdot \pi_3$ is a v-cycle and the minimality of i guarantees that $drop(\pi_1 \cdot \pi_2) \geq drop(\ell) \geq -n$. Moreover, by construction $0 \leq |\pi_1|, |\pi_2|, |\pi_3| \leq |G|$. $\square$

We next consider the problem of deciding whether or not a graph contains a cycle with positive weight.

<u>NoPositiveCycle</u>

INPUT: A weighted graph G.

QUESTION: Are all cycles in G weakly negative?

A naïve approach to this problem would be to enumerate all simple cycles of G and then to check for each of them whether or not they have positive weight. However, such an algorithm would not run in polynomial time since there is potentially an exponential number of simple cycles in a given graph. Instead, Algorithm 1, which is a variant of the celebrated Bellman-Ford algorithm, see *e.g.* [36], avoids this exponential blow-up by using a dynamic programming approach. Let n be the number of vertices of G, for any vertex v of G, Algorithm 1 contains variables $d_v^0, \ldots, d_v^n$, which are all assumed to be initialised with 0. Each d_v^i keeps the maximum weight of some path of length at most i that ends in v. Since the longest simple path in G has length at most $n-1$, $d_v^n > d_v^{n-1}$ for some vertex v indicates that there exists a positive cycle in G. In this case, Algorithm 1 returns false, and true otherwise. Clearly, the running time of the algorithm is polynomial in n.

Lemma 4.1.4 *Given a graph G, Algorithm 1 decides* NoPositiveCycle *in time $O(|G|^2)$.*

We close this section by deriving from Algorithm 1 a sentence in QFPA that decides NoPositiveCycle. To this end, we construct a formula $\varphi(G)$ such that φ is satisfiable if, and only if, G does not contain any positive cycle. The basic idea is that $\varphi(G)$ contains first-order variables d_v^i that represent the same d_v^i from Algorithm 1 and that we can unravel the for-loops from Algorithm 1. Formally, we first ensure that all d_v^0 are initialised with 0:

$$\varphi_0 \stackrel{\text{def}}{=} \bigwedge_{v \in V} d_v^0 = 0$$

Next, we encode the computation of the d_v^i in terms of the previously computed values d_v^{i-1} for each $i \in [n]$, where $n = \#V$. What makes the formula below look slightly

Algorithm 1 Variant of the Bellman-Ford algorithm that returns **true** if, and only if, the graph G does not contain a positive cycle.

```
Input: $G = (V, E, \mu)$
  $n = \#V$
  for $i = 1$ to $n$ do
    for all $v \in V$ do
      $d_v^i := \max\left(\{0\} \cup \{d_u^{i-1} + \mu(u,v) : (u,v) \in E\}\right)$
    end for
  end for
  if there exists $v \in V$ such that $d_v^n > d_v^{n-1}$ then
    return false
  else
    return true
  end if
```

more complicated is the encoding of the maximum function: we first check if the edge we are currently considering results in a maximum value for d_v^i among all other incoming edges. If this is the case, we check if it improves d_v^{i-1} and set d_v^i accordingly.

$$\varphi_i \overset{\text{def}}{=} \bigwedge_{v \in V} \bigwedge_{(u,v) \in E} \left(\bigwedge_{(w,v) \in E} d_w^{i-1} + \mu(w,v) \leq d_u^{i-1} + \mu(u,v) \right) \rightarrow$$
$$\rightarrow \Big(\big(d_u^{i-1} + \mu(u,v) > d_v^{i-1} \rightarrow d_v^i = d_u^{i-1} + \mu(u,v) \big) \wedge$$
$$\wedge \left(d_u^{i-1} + \mu(u,v) \leq d_v^{i-1} \rightarrow d_v^i = d_u^{i-1} \right) \Big)$$

Finally, we need to assert that no d_v^n improves d_v^{n-1}:

$$\varphi_n \overset{\text{def}}{=} \bigwedge_{v \in V} d_v^n \leq d_v^{n-1}$$

We note that $|\varphi_i| = O(|G|^3)$ for each $i \in [0, n-1]$. The QFPA formula $\varphi(G)$ is now obtained by taking the conjunction of φ_0 up to φ_n.

Lemma 4.1.5 *Given a graph G, there exists QFPA formula $\varphi(G)$ such that $\varphi(G)$ is satisfiable if, and only if, G does not contain any positive cycles. Moreover, $|\varphi| = O(|G|^4)$.*

Proof. Let each φ_i be defined as above, it is easily checked that

$$\varphi(G) \stackrel{\text{def}}{=} \exists_{v\in V} d_v^0, \ldots, d_v^n. \bigwedge_{i\in[0,n]} \varphi_i$$

is a QFPA formula with the desired properties. $\square$

4.1.3 Path Flows

In the previous section, we have seen that positive cycle templates allow for providing a polynomial-size certificate for paths of potentially exponential length. We are now going to introduce the concept of *path flows*, which serve a similar purpose. Path flows enable us to encode sets of paths in a graph in a succinct way. Let $G = (V, E, \mu)$ be a fixed graph.

Definition 12 A *flow* is a function $f : E \to \mathbb{N}$. Given $s, t \in V$, we call $f : E \to \mathbb{N}$ an *s-t path flow* if there exists a *corresponding path* π starting in s and ending in t such that for all $e \in E$,

$$f(e) = \#\{i \in \mathbb{N} : e = (\pi(i), \pi(i+1)), i \in [|\pi|]\}.$$

$\Diamond$

Given a path π, the corresponding path flow abstracts away the order in which edges are traversed and only keeps information on how often each edge occurs along π. Since numbers are encoded in binary, we can immediately see that the size to represent a path flow corresponding to π grows logarithmically in $|\pi|$. As a notational convention, we denote by f_π the path flow corresponding to a path π. For an edge $e = (v, w)$ of a graph G, we denote by f_e the v-w path flow for which $f(e) \stackrel{\text{def}}{=} 1$ and $f(e') \stackrel{\text{def}}{=} 0$ for all $e \neq e'$.

The *weight* of a flow f is defined as

$$\mathit{weight}(G, f) \stackrel{\text{def}}{=} \sum_{e\in E} f(e)\mu(e).$$

If G is clear from the context, we just write $\mathit{weight}(f)$ to denote the weight of f. Note that $\mathit{weight}(G, f_\pi) = \mathit{weight}(\pi)$ for any path π.

By $F(f) \overset{\text{def}}{=} \{e : f(e) > 0\}$ we denote the *support of the flow* f. A support can be seen as a template of a path flow selecting the set of edges that a corresponding path traverses. Most of the time, the support of a flow f is clear from the context and for convenience we denote it by F. We call $F \subseteq E$ an s-t support if the subgraph $G/(F \cup \{(t, s)\})$ is connected.

An alternative characterisation of path flows in terms of Eulerian path conditions is provided by the following lemma. In the following, let $in(v) \overset{\text{def}}{=} \{w : (w, v) \in E\}$ and $out(v) \overset{\text{def}}{=} \{w : (v, w) \in E\}$ denote the set of *incoming* respectively *outgoing vertices of* v for any $v \in V$.

Lemma 4.1.6 *A flow f is an s-t path flow, if and only if, f satisfies the following conditions:*

(i) (a) If $s = t$ then

$$\sum_{w \in out(v)} f(v, w) = \sum_{w \in in(v)} f(w, v) \qquad \text{for all } v \in V. \tag{4.1}$$

(b) If $s \neq t$ then

$$\sum_{w \in out(v)} f(v, w) = \sum_{w \in in(v)} f(w, v) \qquad \text{for all } v \in V \setminus \{s, t\}, \tag{4.2}$$

$$\sum_{w \in out(s)} f(s, w) = \sum_{w \in in(v)} f(w, s) + 1, \tag{4.3}$$

$$\sum_{w \in out(t)} f(t, w) = \sum_{w \in in(t)} f(w, t) - 1. \tag{4.4}$$

(ii) $F(f)$ is an s-t support.

Proof. The proof is by induction on $n = \sum_{e \in E} f(e)$. For the induction step, suppose the lemma holds for all $k < n$.

("$\Rightarrow$") Suppose that f comes from a path $\pi = \pi' \cdot vt$ whose length is n. Let f' be the s-v path flow corresponding π'. By the induction hypothesis, (i) and (ii) hold for f', and we have $f = f'[(v, t) \mapsto f'(v, t) + 1]$. We only consider the case $s \neq t$ and $v \neq s$, the remaining cases follow along similar arguments. It is not difficult to check that $G/(F' \cup \{(v,t),(t,s)\})$ is connected. Moreover, $\sum_{w \in out(s)} f(s,w) = \sum_{w \in out(s)} f'(s,w) = \sum_{w \in in(s)} f'(w,s) + 1 = \sum_{w \in in(s))\in E} f(w,s) + 1$. For v, we have $\sum_{w \in out(v,w)} f(v,w) = \sum_{w \in out(v)} f'(v,w) + 1 = \sum_{w \in in(v)} f'(w,v) = \sum_{w \in in(v)} f(w,v)$. Finally, $\sum_{w \in out(t)} f(t,w) = \sum_{w \in out(t,w)} f'(t,w) = \sum_{w \in in(t)} f'(w,t) = \sum_{w \in in(t)} f(w,t) - 1$. It is easily checked that (4.2) holds for the remaining vertices.

("$\Leftarrow$") Let f be an s-t path flow. Choose $v \in in(t)$ such that $f(v,t) > 0$. Set $f' = f[(v,t) \mapsto f(v,t) - 1]$. In the following, we assume $s \neq t$ and $v \neq s$, the other cases follow similarly. We claim that f' is an s-v path flow that fulfills the conditions $(i)(b)$ and (ii). By the induction hypothesis, it then follows that there exists an s-v path. Indeed, it is easy to check that (ii) holds. Moreover, $\sum_{w \in out(s)} f'(s,w) = \sum_{w \in out(s)} f(s,w) = \sum_{w \in in(s) \in E} f(w,s) + 1 = \sum_{w \in in(s)} f'(w,s) + 1$; $\sum_{w \in out(v)} f'(v,w) = \sum_{w \in out(v) \in E} f(v,w) - 1 = \sum_{w \in in(w,v)} f(w,v) - 1 = \sum_{w \in in(v)} f'(w,v) - 1$; and $\sum_{w \in out(t)} f'(t,w) = \sum_{w \in out(t)} f(t,w) = \sum_{w \in in(t)} f(w,t) - 1 = \sum_{w \in in(t)} f'(w,t)$. $\square$

In the proof of the "only if" direction of the previous lemma, we choose an arbitrary incoming vertex v of t with $f(v,t) > 0$. This non-determinism can be seen as the cause why in general a path flow does not uniquely determine a path. The benefit of the characterisation in terms of Eulerian path flow conditions is that it allows for rephrasing the existence of a path flow with a certain weight into a sentence in QFPA.

Lemma 4.1.7 *Let $s, t \in V$, $n, n' \in \mathbb{N}$ and F be an s-t support, there exists a QFPA formula $\varphi(G, F, s, t)(c, c')$ such that $\varphi[n/c, n'/c']$ if, and only if, there exists an s-t path flow f with support F and weight$(f) = n' - n$. Moreover, $|\varphi| = O(|G|^2)$.*

Proof. First, we observe that if we treat each $f(e)$ as a first-order variable in the Eulerian path flow conditions in Lemma 4.1.6, each condition (4.1)–(4.4) yields a QFPA formula open in $f(e_1), \ldots, f(e_k)$. Denote by $\psi(f(e_1), \ldots, f(e_k))$ the appropriate conjunction of QFPA formulae derived from these conditions depending on whether or not $s = t$. We have $\psi = O(|G|^2)$.

Next, we need an additional formula fixing the weight of f and ensuring that all edges not in F have zero flow:

$$\psi'(f(e_1), \ldots, f(e_k)) \overset{\text{def}}{=} \sum_{e \in F} f(e)\mu(e) = c' - c \wedge \bigwedge_{e \in E \setminus F} f(e) = 0.$$

We set $\varphi(G, F, s, t) \overset{\text{def}}{=} \exists_{e \in E} f(e).\psi \wedge \psi'$, which by Lemma 4.1.6 has the desired properties. □

A nice property of path flows is that they are additive. Given flows f, f', we define

$$f + f' \overset{\text{def}}{=} e \mapsto f(e) + f'(e).$$

Addition of path flows can be seen as the operation corresponding to concatenation of paths.

Lemma 4.1.8 *Let f, f' be paths flows, then $f + f'$ is an s-t path flow if there exists $v \in V$ such that*

(i) f is an s-v path flow and f' is an v-t path flow; or

(ii) f is an s-t path flow, f' is a v-v path flow and $F \cup F'$ is an s-t support.

Proof. The lemma follows in both cases from a straightforward application of Lemma 4.1.6. □

In the remainder of this section, we look at different ways to decompose path flows into sequences of path flows. Those decompositions are later going to be used for our results on reachability in one-counter automata. We first show that any path flow corresponding to a cycle can be decomposed into path flows whose supports induce subgraphs that are loops.

Lemma 4.1.9 *A flow f is a v-v path flow if, and only if, there are path flows $f_1, \ldots, f_j, j \in [|G|]$ such that each G/F_i is a loop and $f = \sum_{i \in [j]} f_i$.*

Proof. ("$\Rightarrow$") We show the lemma by induction on the number of edges in G/F. For the induction step, choose some subset of edges E' of the edges of G/F such that G/E' is a loop and there is some $e' \in E'$ such that $f(e')$ is minimal among all edges from F. Define f_1 such that $f_1(e) \overset{\text{def}}{=} f(e')$ if $e \in E'$ and $f_1(e) \overset{\text{def}}{=} 0$ otherwise. By definition, G/F_1 is a cycle. Moreover, let f' be a flow such that $f = f_1 + f'$, such a flow exists due to our choice of e'. Now f' is not necessarily a path flow, since G/F' may consist of several disjoint strongly connected components. However, by restricting f' to each of these strongly connected components and by applying the induction hypothesis on each of these restricted flows, we obtain the required path flows $f_2, \ldots, f_j$ that give $f = \sum_{i \in [j]} f_i$.

("$\Leftarrow$") It is easily checked that the conditions $(i)(a)$ and (ii) in Lemma 4.1.6 are fulfilled for f. Hence f is a v-v path flow for some vertex v of G/F. $\square$

Using the previous lemma, we now show that an arbitrary path flow can be decomposed into a path flow whose support induces a subgraph that is a simple path and a number of path flows whose supports induce subgraphs that are simple cycles.

Lemma 4.1.10 *A flow f is an s-t path flow if, and only if, there are j path flows f_i with $j = O(|G|^2)$ and a path flow f_0 such that $f = f_0 + \sum_{i \in [j]} f_i$, G/F_0 is a simple s-t path, each G/F_i is a simple cycle, and $G/(\bigcup_{i \in [0,j]} F_i)$ is connected.*

Proof. If $s = t$ then the lemma directly follows from Lemma 4.1.9. Thus, we subsequently assume $s \neq t$.

("$\Rightarrow$") Let π be a path corresponding to f, and let π_0 be obtained from π by deleting all cycles. Define f_0 such that $f_0(e) \overset{\text{def}}{=} 1$ if $e \in \mathit{edges}(\pi_0)$ and $f_0(e) \overset{\text{def}}{=} 0$ otherwise. Let f' be the flow such that $f = f' + f_0$, which is not necessary a path flow. However, by successively restricting f' to each of the strongly connected components in G/F' and applying Lemma 4.1.9 to each component, we obtain the required path flows f_i.

("$\Leftarrow$") It is easily checked that the conditions $(i)(b)$ and (ii) in Lemma 4.1.6 are fulfilled for f. Hence f is an s-t path flow. $\square$

We close this section by introducing *edge decompositions* of path flows. An edge decomposition of a path flow f is a sequence of tuples $(f_i, e_i)_{i\in[m]}$ consisting of a path flow and an edge such that the edge e_i has zero flow in any f_j for $j > i$ and all the components of the decomposition sum up to f. By sorting the edges a path π traverses in the order of their last appearance in π, each path π gives rise to a canonical edge decomposition of f_π. We first define edge decompositions for supports and then for path flows.

Definition 13 Given an s-t support F, a *support-edge decomposition of F* is a sequence of tuples $(F_i, v_i, w_i, e_i)_{i\in[m]}$ with $F_i \subseteq F$, $v_1 = s$, $v_{m+1} \overset{\text{def}}{=} t$ such that

- F_i is a v_i-w_i support, $e_i = (w_i, v_{i+1})$, $i \in [m]$,
- $e_i \notin F_j$ for all $1 \le i < j \le m$,
- $F = \bigcup_{i\in[m]}\{e_i\}$.

An *edge decomposition* of a path flow f is a sequence of tuples $(f_i, e_i)_{i\in[m]}$, where each f_i is v_i-w_i path flow, $v_1 = s$, $v_{m+1} = t$ and $e_i = (w_i, v_{i+1})$ such that $f = \sum_{i\in[m]} f_i + f_{e_i}$ and $(F_i, v_i, w_i, e_i)_{i\in[m]}$ is a support-edge decomposition. $\Diamond$

Figure 4.2 gives an example of an edge decomposition of a path flow. The path flow f is decomposed into $(f_i, e_i)_{i\in[4]}$ and it is easily verified that $f = \sum_{i\in[4]} f_i + f_{e_i}$. An example of a path inducing this edge decomposition is a path that traverses the following edges in this order:

$$\underbrace{e_1e_1e_3e_4e_2e_1e_1e_1}_{f_1}\, e_1\, \underbrace{e_3e_4e_4e_2e_3e_4}_{f_2}\, e_2e_3\, \underbrace{e_4e_4e_4}_{f_4}\, e_4.$$

4.1.4 The NP Upper Bound

Using the concepts introduced and developed in the previous sections, we are now going to show that reachability in one-counter automata is in NP.

In the first part of this section, we will only consider one-counter automata *without* zero tests. This allows us to view one-counter automata as weighted graphs. For

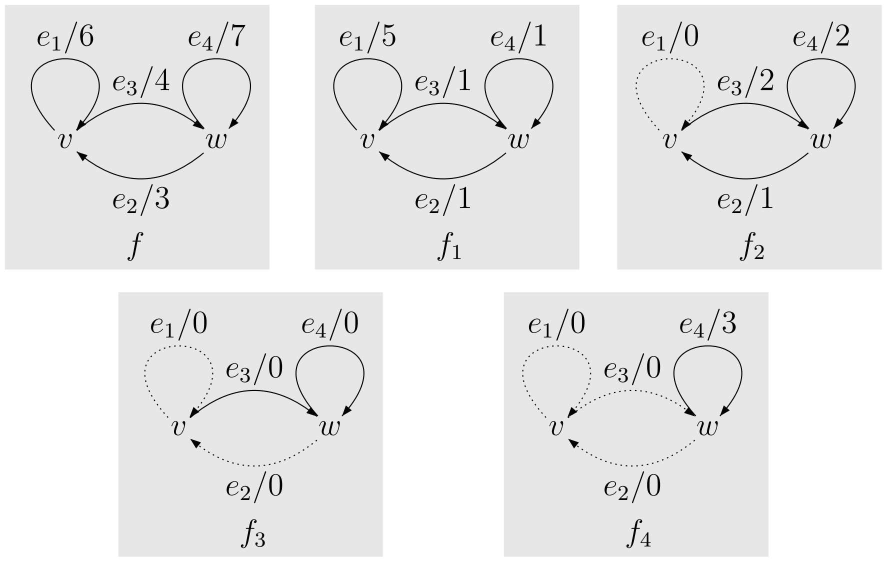

Figure 4.2: An example of an edge decomposition of a path flow. Each path flow is determined by the number next to the edges of each graph, *e.g.*, $f(e_2) = 3$. Here, the decomposition of the path flow f is given by $(f_1, e_1)(f_2, e_2)(f_3, e_3)(f_4, e_4)$.

now, let us fix a zero-test free one-counter automaton $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda, \xi)$. The *weighted graph corresponding to* $\mathcal{A}$ is $G_{\mathcal{A}} \overset{\text{def}}{=} (Q, \Delta, \lambda)$. Just as we can relate $\mathcal{A}$ with $G_{\mathcal{A}}$, we can relate runs in $T(\mathcal{A})$ with paths in $G_{\mathcal{A}}$: the *path corresponding to a run* $(q_1, c_1)(q_2, c_2) \cdots (q_n, c_n)$ in $T(\mathcal{A})$ is the path $q_1 q_2 \ldots q_n$ in $G_{\mathcal{A}}$, *i.e.*, the projection on the control locations visited.

However, the converse does not hold in general: a path in $G_{\mathcal{A}}$ does not necessarily correspond to a run between two configurations in $T(\mathcal{A})$. Informally speaking, if the counter value we start with is not large enough, a path in G might force the counter to drop below zero. The following lemma provides the connection between paths in $G_{\mathcal{A}}$ and runs in $T(\mathcal{A})$.

Lemma 4.1.11 *Let π be a q-q' path in $G_{\mathcal{A}}$ and $n, n' \in \mathbb{N}$. There is a run $(q, n) \to_{\mathcal{A}}^{*} (q', n')$ that π corresponds to if, and only if, $drop(\pi) \geq -n$ and $weight(\pi) = n' - n$.*

Proof. We show the statement by induction on $|\pi|$.

("$\Rightarrow$") Suppose $\varrho = \varrho' \cdot (q'', n'')(q', n')$ is a run that π corresponds to for some $n'' \in \mathbb{N}$. By the induction hypothesis, $drop(\pi') \geq -n$ and $weight(\pi') = n'' - n$, where π' is the path corresponding to ϱ'. Moreover, $n' = n'' + \mu(q'', q')$, hence

$\mathit{weight}(\pi) = \mathit{weight}(\pi') + \mu(q'', q') = n'' - n + \mu(q', q'') = n' - n$ and $\mathit{drop}(\pi) = \min\{\mathit{drop}(\pi'), \mathit{weight}(\pi)\} \geq -n$.

("$\Leftarrow$") Let $\pi = \pi' \cdot q''q'$ be a path in $G_{\mathcal{A}}$ such that $\mathit{drop}(\pi) \geq -n$ and $\mathit{weight}(\pi) = n' - n$. By the induction hypothesis, there is a run $\varrho' : (q, n) \to_{\mathcal{A}}^{*} (q'', n'')$, where $n'' = \mathit{weight}(\pi') + n$. As $n' = n'' + \mu(q, q') \geq 0$, $\varrho \overset{\text{def}}{=} \varrho' \cdot (q'', n'')(q', n')$ is the desired run. □

As already discussed earlier, a path witnessing reachability might become exponential in the size of $\mathcal{A}$ and is for that reason unsuitable for providing an NP upper bound. However, the size of a path flow corresponding to a witnessing path π is logarithmic in the size of π and hence can be guessed in NP. The only drawback is that given a path flow we cannot reconstruct a witnessing path, and thus we cannot be sure whether a given path flow is induced by a witnessing path. For that reason, given two configurations in $T(\mathcal{A})$, we have to look for suitable conditions on path flows that guarantee the existence of a path witnessing reachability. We call these conditions *reachability criteria*.

Definition 14 Let G be a graph, f an s-t path flow and $n, n' \in \mathbb{N}$. Then (G, f, n, n') fulfills the

(i) *type-1 reachability criteria* if

- G/F does not contain positive cycles
- $\mathit{weight}(f) = n' - n$
- f has an edge decomposition $(f_i, e_i)_{i \in [m]}$ such that $\sum_{i \in [j]} \mathit{weight}(f_i + f_{e_i}) \geq -n$ for all $j \in [m]$;

(ii) *type-2 reachability criteria* if

- $(G^{\mathsf{op}}, f^{\mathsf{op}}, n', n)$ fulfills the type-1 reachability criteria;

(iii) *type-3 reachability criteria* if

- $\mathit{weight}(f) = n' - n$

- there is a positive s-cycle template ℓ in G with respect to n
- there is a positive t-cycle template ℓ' in G^{op} with respect to n'. $\Diamond$

We call (G, f, n, n') a *type-i reachability certificate* if (G, f, n, n') fulfills the type-i reachability criteria.

Our aim now is to show that given counter values n, n' and a q-q' path flow f, if $(G_{\mathcal{A}}, f, n, n')$ is a reachability certificate then (q', n') is reachable from (q, n) in $T(\mathcal{A})$. Before we begin with the formal part, let us explain on an intuitive level why this is the case.

Suppose that (G, f, n, n') is a type-1 reachability certificate. Since G/F does not contain any positive cycles, the weight of a path that corresponds to f always decreases whenever it repeatedly traverses an edge. The constraints on the edge decomposition of f make sure that the last time we traverse an edge the weight of a corresponding path does not go below $-n$. Hence for any fixed edge, at any time we traverse it the weight of the current path segment is above $-n$. Since the edge decomposition ranges over all edges that have flow greater than zero, it is guaranteed that a path exists that fulfils the conditions from Lemma 4.1.11. The case of type-2 reachability certificates reduces to the type-1 case. The definition of a type-3 reachability certificate can be read as requiring the existence of a suitable path flow f together with a suitable positive and negative cycle at the source s respectively target t of f. The positive cycle at s whose drop is above $-n$ guarantees that starting from a configuration (q, n) we can reach a configuration (q, m') such that $m' > m$ for any $m > n$. If π is some path that f corresponds to, we can thus "pump up" the counter value as high as we need in order to ensure that starting from this counter value π does not force the counter to drop below 0. Once some configuration (t, m'') is reached, we can use the negative cycle at t to bring the counter value down to n'. Consequently, we can reach (t, n') from (q, n) in $T(\mathcal{A})$. The following lemma makes our intuition formal.

Lemma 4.1.12 *Let (q, n) and (q', n') be configurations of a one-counter automaton $\mathcal{A}$, $G_{\mathcal{A}}$ the graph corresponding to $\mathcal{A}$ and f a q-q' path flow. We have that* *(i) if $(G_{\mathcal{A}}, f, n, n')$ is a type-1 reachability certificate;*

(ii) if $(G_{\mathcal{A}}, f, n, n')$ is a type-2 reachability certificate; or

(iii) if $(G_{\mathcal{A}}, f, n, n')$ is a type-3 reachability certificate

then $f = f_\pi$ for some path π corresponding to a run $\varrho : (q, n) \rightarrow^*_{\mathcal{A}} (q', n')$ in $T(\mathcal{A})$.

Proof. (i) Let $(f_i, e_i)_{i\in[m]}$ be the edge decomposition of f from Definition 14(i), where we identify e_i as (v_i, w_i). For each f_i choose some arbitrary path π_i such that $f_i = f_{\pi_i}$ for $i \in [m]$ and set $\pi \overset{\text{def}}{=} \pi_1 \cdot v_1 w_1 \cdot \pi_2 \cdot v_2 w_2 \cdot \ldots \cdot \pi_m \cdot v_m w_m$. By assumption, G/F does not contain a positive cycle and consequently there is no positive cycle in π. Hence for two prefixes $\pi_1' \cdot v_j w_j$ and $\pi_2' \cdot v_j w_j$ of π with $|\pi_1| \le |\pi_2|$ that traverse the same last edge, we have $\mathit{weight}(\pi_1' \cdot v_j w_j) \ge \mathit{weight}(\pi_2' \cdot v_j w_j)$. It follows that we can obtain the drop of π by just considering the segments of π in which each edge is visited the last time. We deduce that

$$\begin{aligned}
\mathit{drop}(\pi) &= \min\{\mathit{weight}(\pi_1 \cdot v_1 w_1 \cdot \ldots \cdot \pi_j \cdot v_j w_j) : j \in [m]\} \\
&= \min\left\{\sum_{i\in[j]} \mathit{weight}(\pi_i \cdot v_j w_j) : j \in [m]\right\} \\
&= \min\left\{\sum_{i\in[j]} \mathit{weight}(f_i + f_{e_i}) : j \in [m]\right\} \\
&\ge -n.
\end{aligned}$$

The application of Lemma 4.1.11 yields that the desired run $\varrho : (q, n) \rightarrow^*_{\mathcal{A}} (q', n')$ in $T(\mathcal{A})$ exists.

(ii) By (i), we have that there exists a path π^{op} in G^{op} such that $\mathit{weight}(\pi^{\mathsf{op}}) = n - n'$ and $\mathit{drop}(\pi^{\mathsf{op}}) \ge -n'$. Using Lemma 4.1.2(iii), it follows that

$$\begin{aligned}
&\mathit{drop}(\pi^{\mathsf{op}}) \ge -n' \\
\iff &\mathit{drop}(\pi) - \mathit{weight}(\pi) \ge -n' \\
\iff &\mathit{drop}(\pi) - n' + n \ge -n' \\
\iff &\mathit{drop}(\pi) \ge -n.
\end{aligned}$$

By applying Lemma 4.1.11, it follows that a desired run $\varrho : (q, n) \to_{\mathcal{A}}^{*} (q', n')$ in $T(\mathcal{A})$ exists.

(iii) Let π be some path with the corresponding flow f, and let ℓ be the positive s-cycle template with respect to n and ℓ' the positive t-cycle template in G^{op} with respect to n'. By Lemma 4.1.3, ℓ induces a positive v-cycle ℓ_1 with $\mathit{drop}(\ell_1) \geq -n$, and ℓ' induces a positive ℓ_2 in G^{op} such that $\mathit{drop}(\ell_2) \geq -n'$. We use ℓ_1 and ℓ_2^{op} in order to appropriately "pump up" and "pump down" π. Let $m = \mathit{weight}(\ell_1)$ and $m' = \mathit{weight}(\ell_2)$. Choose $a \in \mathbb{N}$ such that $a \cdot m \cdot m' \geq \mathit{drop}(\pi)$ and define $\pi' = {\ell_1}^{a \cdot m'} \cdot \pi \cdot ({\ell_2}^{a \cdot m})^{\mathsf{op}}$. Clearly, we have $\mathit{weight}(\pi') = \mathit{weight}({\ell_1}^{a \cdot m'}) + \mathit{weight}(\pi) + \mathit{weight}(({\ell_2}^{a \cdot m})^{\mathsf{op}}) = \mathit{weight}(\pi)$. Thus, it remains to show that $\mathit{drop}(\pi') \geq -n$, which allows us to apply Lemma 4.1.11 and to conclude that π' has corresponding run $\varrho : (q, n) \to_{\mathcal{A}}^{*} (q', n')$. Subsequently, we make implicit use of the statements from Lemma 4.1.2. We have that

$$\begin{aligned}
& \mathit{drop}(\pi') \geq -n \\
\iff & \mathit{drop}({\ell_1}^{a \cdot m'} \cdot \pi) \geq -n \text{ and } \mathit{weight}({\ell_1}^{a \cdot m'} \cdot \pi) + \mathit{drop}(({\ell_2}^{a \cdot m})^{\mathsf{op}}) \geq -n \\
\iff & \mathit{drop}({\ell_1}^{a \cdot m'}) \geq -n \text{ and } \mathit{weight}({\ell_1}^{a \cdot m'}) + \mathit{drop}(\pi) \geq -n \text{ and} \\
& \mathit{weight}({\ell_1}^{a \cdot m'} \cdot \pi) + \mathit{drop}(({\ell_2}^{a \cdot m})^{\mathsf{op}}) \geq -n.
\end{aligned}$$

By Lemma 4.1.2(ii) and by the choice of a, we have $\mathit{drop}({\ell_1}^{a \cdot m'}) \geq -n$ and $\mathit{weight}({\ell_1}^{a \cdot m'}) + \mathit{drop}(\pi) \geq -n$. Thus, it remains to show that $\mathit{weight}({\ell_1}^{a \cdot m'} \cdot \pi) + \mathit{drop}(({\ell_2}^{a \cdot m})^{\mathsf{op}}) \geq -n$. It follows that

$$\begin{aligned}
& \mathit{weight}({\ell_1}^{a \cdot m'} \cdot \pi) + \mathit{drop}(({\ell_2}^{a \cdot m})^{\mathsf{op}}) \geq -n \\
\iff & \mathit{weight}({\ell_1}^{a \cdot m'} \cdot \pi) + \mathit{drop}({\ell_2}^{a \cdot m}) - \mathit{weight}({\ell_2}^{a \cdot m}) \geq -n \\
\iff & \mathit{drop}({\ell_2}^{a \cdot m}) + \mathit{weight}(\pi) \geq -n \\
\iff & \mathit{drop}({\ell_2}^{a \cdot m}) + n' - n \geq -n \\
\iff & \mathit{drop}({\ell_2}^{a \cdot m}) \geq -n'
\end{aligned}$$

By assumption, $\mathit{drop}(\ell_2) \geq -n'$ and hence by Lemma 4.1.2(ii) we get $\mathit{drop}({\ell_2}^{a \cdot m}) \geq -n'$. □

Thus, any reachability certificate guarantees the existence of a path in $G_{\mathcal{A}}$ that corresponds to a run. However, the converse direction does not hold. An arbitrary run does in general not yield a path flow that fulfills some reachability criterion. As we are now going to show, it is however possible, starting from an arbitrary run, to construct a run that decomposes into at most three runs that all yield path flows that give reachability certificates. Before showing this fact, we need to prove the following technical lemma.

Lemma 4.1.13 *Let $\varrho : (q,n) \to_{\mathcal{A}}^{*} (q',n')$ be a run in $T(\mathcal{A})$ with the corresponding path π in $G_{\mathcal{A}}$ and let F be the support of f_π. If π does not contain any positive cycle then either $G_{\mathcal{A}}/F$ does not contain any positive cycles or there is a path π' in $G_{\mathcal{A}}$ that factors as $\pi' = \pi_1 \cdot \pi_2 \cdot \pi_3$ and corresponds to a run $\varrho' : (q,n) \to_{\mathcal{A}}^{*} (q',n')$ in $T(\mathcal{A})$ such that $|\pi_1| < |\pi|$ and π_2 is a positive cycle.*

Proof. Suppose that $G_{\mathcal{A}}/F$ contains a positive cycle ℓ. Let p be the first vertex of ℓ that occurs in π, and let $m \in \mathbb{N}$ be such that the configuration (p,m) is first reached by ϱ. We claim that there is a positive cycle at p in $G_{\mathcal{A}}$ that corresponds to a run $(p,m) \to_{\mathcal{A}}^{*} (p,m')$ in $T(\mathcal{A})$ for some $m' > m$.

If ℓ does not correspond to such a run starting from (p,m) we argue as follows. Factor ℓ as $\ell = \pi_1' \cdot \pi_2'$ with $\pi_1' : p \to_{G_{\mathcal{A}}}^{*} r$, $\pi_2' : r \to_{G_{\mathcal{A}}}^{*} p$ such that r is the node with the maximum decrement in ℓ, *i.e.*, $\mathit{weight}(\pi_1') = \mathit{drop}(\ell)$ and whence $\mathit{weight}(\pi_1') < -m$. Since p is the first vertex of ℓ visited by π, r is visited by π sometime after the first visit of p. So there is a p-r path π_3' in $G_{\mathcal{A}}$ such that $\mathit{weight}(\pi_3') \geq \mathit{drop}(\pi_3') \geq -m > \mathit{weight}(\pi_1')$. Consider now the cycle $\ell' \overset{\text{def}}{=} \pi_3' \cdot \pi_2'$. It follows that ℓ' is a positive cycle, since

$$\begin{aligned} \mathit{weight}(\ell') &= \mathit{weight}(\pi_3') + \mathit{weight}(\pi_2') \\ &> \mathit{weight}(\pi_1') + \mathit{weight}(\pi_2') \\ &= \mathit{weight}(\ell). \end{aligned}$$

Moreover, by our choice of r we have $drop(\pi'_2) \geq 0$ and hence we conclude that

$$\begin{aligned} drop(\ell') &\geq drop(\pi'_3) + drop(\pi'_2) \\ &\geq -m + 0 \\ &= -m. \end{aligned}$$

Hence, Lemma 4.1.11 implies that ℓ' corresponds to a run from $(p, m) \rightarrow^*_{\mathcal{A}} (p, m')$ in $T(\mathcal{A})$.

Next we observe that the first occurrence of p in π actually lies on a negative cycle in π. This is because π must visit p in π again, otherwise ℓ would not exist in f_π. By assumption all cycles in π are negative. Thus, we can decompose ϱ as $\varrho_1 \cdot \varrho_2 \cdot \varrho_3$ with $\varrho_1 : (q, n) \rightarrow^*_{\mathcal{A}} (p, m)$, $\varrho_2 : (p, m) \rightarrow^*_{\mathcal{A}} (p, m'')$ and $\varrho_3 : (p, m'') \rightarrow^*_{\mathcal{A}} (q', n')$ with $m'' < m$. Denote by π''_1, π''_2 and π''_3 the path corresponding to ϱ_1, ϱ_2 and ϱ_3 respectively.

In order to obtain the path $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ required in the lemma, we reuse an idea from the proof of Lemma 4.1.12(iii). Let $i = weight(\ell')$ and $j = |weight(\pi''_2)|$. Define $\pi_1 = \pi''_1$, $\pi_2 = (\ell')^j$ and $\pi'_3 = \pi_2''^{i+1} \cdot \pi''_3$. Clearly, $|\pi_1| < \pi$ and π_2 is a positive cycle as required. Since the positive cycle $(\ell')^j$ is canceled out by the negative cycle $(\pi''_2)^i$, and by applying Lemma 4.1.11, the required run ϱ' exists. $\square$

We can now use this lemma in order to show that if $(q, n) \rightarrow^*_{\mathcal{A}} (q', n')$ then there exists a run that can be decomposed into three components whose corresponding paths each yield reachability certificates.

Lemma 4.1.14 *There is a run $\varrho : (q, n) \rightarrow^*_{\mathcal{A}} (q', n')$ in $T(\mathcal{A})$ if, and only if, there is a q-q' path π in $G_{\mathcal{A}}$ that can be written as $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ such that there are $n_1, n_2 \in \mathbb{N}$ such that*

- *if $|\pi_1| > 0$ then $(G_{\mathcal{A}}, f_{\pi_1}, n, n_1)$ is a type-1 reachability certificate;*
- *if $|\pi_2| > 0$ then $(G_{\mathcal{A}}, f_{\pi_2}, n_1, n_2)$ is a type-3 reachability certificate; and*
- *if $|\pi_3| > 0$ then $(G_{\mathcal{A}}, f_{\pi_3}, n_2, n')$ is a type-2 reachability certificate.*

Proof. ("$\Rightarrow$") If $\varrho : (q, n) \to_{\mathcal{A}}^{*} (q', n')$ is a run with a corresponding path π whose corresponding path flow has support F such that $G_{\mathcal{A}}/F$ does not contain any positive cycle then π induces a unique vertex decomposition and hence (G, f_{π}, n, n') fulfills the type-1 reachability criteria.

Otherwise, let π' be a path in $G_{\mathcal{A}}$ corresponding to some run $\varrho : (q, n) \to_{\mathcal{A}}^{*} (q', n')$. By repeatedly applying Lemma 4.1.13 to π', we can obtain $\pi_1 \cdot \pi_2' \cdot \pi_3'$ from π' such that $\pi_1 : q \to_{G_{\mathcal{A}}}^{*} p$, $G_{\mathcal{A}}/F_{\pi_1}$ does not contain any positive cycles and $\pi'' = \pi_2' \cdot \pi_3'$ is a p-q' path with π_2' being a positive cycle. If $G_{\mathcal{A}}/F_{\pi''}$ does not contain any negative cycles, by setting $\pi_3 = \pi''$ and $n_1 = n + \mathit{weight}(\pi_1)$, we can easily see that $(G_{\mathcal{A}}, f_{\pi_1}, n, n_1)$ and $(G_{\mathcal{A}}, f_{\pi_3}, n_1, n')$ are type-1 respectively type-2 reachability certificates. It follows that $\pi = \pi_1 \cdot \pi_3$ is the required path.

Otherwise, by repeatedly applying Lemma 4.1.13 to $(\pi'')^{\mathsf{op}}$ in $G_{\mathcal{A}}^{\mathsf{op}}$ we obtain a path that decomposes into $\pi_3^{\mathsf{op}} \cdot (\pi_2'')^{\mathsf{op}} \cdot (\pi_1'')^{\mathsf{op}}$ such that $\pi_2 = \pi_1'' \cdot \pi_2''$ is a p-p' path with $(\pi_2'')^{\mathsf{op}}$ being a positive cycle in $G_{\mathcal{A}}^{\mathsf{op}}$ and $G_{\mathcal{A}}^{\mathsf{op}}/F_{\pi_3^{\mathsf{op}}}$ does not contain any positive cycle. Let $n_1 = n + \mathit{weight}(\pi_1)$ and $n_2 = n' - \mathit{weight}(\pi_3)$. Both π_2' and π_2'' witness the existence of a positive cycle in $G_{\mathcal{A}}$ respectively $G_{\mathcal{A}}^{\mathsf{op}}$ with $\mathit{drop}(\pi_2') \geq -n_1$ and $\mathit{drop}((\pi_2'')^{\mathsf{op}}) \geq -n_2$. Thus, $(G_{\mathcal{A}}, f_{\pi_2}, n_1, n_2)$ fulfills the type-3 reachability criteria. As above, $(G_{\mathcal{A}}, f_{\pi_1}, n, n_1)$ and $(G_{\mathcal{A}}, f_{\pi_3}, n_2, n')$ fulfill the type-1 respectively type-2 reachability certificates, and hence $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ is the required path.

("$\Leftarrow$") This direction follows by combining the statements from Lemma 4.1.12. $\square$

We have thus shown that deciding reachability in $T(\mathcal{A})$ can be reduced to checking for the existence of at most three reachability certificates in $G_{\mathcal{A}}$. We now proceed by showing that checking the existence of a reachability certificate can be phrased in terms of an open formula in QFPA. To begin with, we take the conditions required by the type-1 reachability criteria in Definition 14 and translate them into an open formula in QFPA for a fixed edge decomposition. Suppose we are given G, vertices s, t, a support F and a support edge decomposition $(F_i, v_i, v_i', e_i)_{i \in [m]}$. For $i \in [m]$, let $\varphi(G, F_i, v_i, v_i')(c_i, c_i')$ be the path flow formulae from Lemma 4.1.7 such that $\varphi_i[n/c_i, n'/c_i']$ if, and only if, there exists a v_i-v_i' path flow f_i in G with support F_i such that $\mathit{weight}(f_i) = n' - n$. Furthermore, let $\varphi(G/F)$ be the formula from Lemma

4.1.5 guaranteeing that there are no positive cycles in G/F. The type-1 reachability criteria for a fixed edge decomposition can now be expressed by the following QFPA formula $\varphi(G, F, (F_i, v_i, v_i', e_i)_{i\in[m]})(c, c')$, which is computable in polynomial time:

$$\varphi \overset{\text{def}}{=} \exists_{i\in[m]} c_i, c_i'.\quad \underbrace{\varphi(G/F)}_{\text{no positive cycles}} \quad\wedge\quad \underbrace{\bigwedge_{i\in[m]} \varphi(G, F_i, v_i, v_i')(c_i, c_i')}_{\text{there are path flows } f_i \text{ with weight } c_i' - c_i} \quad\wedge$$
$$\wedge \quad \underbrace{\bigwedge_{i\in[m]} \sum_{j\in[i]} c_i' - c_i + \mu(e_i) \geq -c}_{\text{weights of the edge decomposition sum up correctly}} \quad \wedge \underbrace{\sum_{i\in[m]} c_i' - c_i + \mu(e_i) = c' - c}_{\text{total weight matches}} \qquad (4.5)$$

We can now prove the following lemma.

Lemma 4.1.15 *Given a graph G and vertices s, t, the sets*

$\{(n, n') :$ *there exists an s-t path flow f such that (G, f, n, n') is a type-1 reachability certificate*$\}$

$\{(n, n') :$ *there exists an s-t path flow f such that (G, f, n, n') is a type-2 reachability certificate*$\}$

are definable via sets $R_1(G, s, t)$ and $R_2(G, s, t)$ of QFPA formulae, where $|\varphi| = O(|G|^4)$ for each $\varphi \in R_1(G, s, t) \cup R_2(G, s, t)$.

Proof. For each possible s-t support F and each support edge decomposition $(F_i, v_i, v_i', e_i)_{i\in[m]}$ of F, $R_1(G, s, t)$ is the smallest set containing a formula $\varphi(G, F, (F_i, v_i, v_i', e_i)_{i\in[m]})$ as in Equation (4.5). As $|\varphi(G/F)| = O(|G|^4)$ dominates every other conjunct and by combining Lemmas 4.1.5 and 4.1.7, it is easily checked that R_1 has the desired properties. The set R_2 can be defined as $R_2(G, s, t) \overset{\text{def}}{=} R_1(G^{\mathsf{op}}, t, s)$. □

We now proceed by proving a similar statement for type-3 reachability certificates. First, we show how to express the conditions on positive cycle templates in QFPA. Recall that a v-cycle ℓ is a positive v-cycle template with respect to n if it factors into $\pi_1 \cdot \pi_2 \cdot \pi_3$ such that $drop(\pi_1 \cdot \pi_2) \geq -n$ and $weight(\pi_2) > 0$. Define the QFPA formula $\varphi(G, \pi_1, \pi_2)$ as follows, where φ_d and φ_w are the QFPA formulae defining the drop and the weight of a path as defined in Lemma 4.1.1:

$$\varphi(G, \pi_1, \pi_2)(c) \stackrel{\text{def}}{=} \exists d, d'.(\varphi_d(G, \pi_1 \cdot \pi_2)(d, d') \wedge d' - d \geq -c)\wedge \\ \wedge \exists d, d'.(\varphi_w(G, \pi_2)(d, d') \wedge d' - d > 0) \quad (4.6)$$

It follows from Lemma 4.1.1 that $|\varphi(G, \pi_1, \pi_2)(c)| = O(|G|^2)$. By applying the same lemma it is easily checked that $\varphi(G, \pi_1, \pi_2)[n/c]$ holds if, and only if, $\ell = \pi_1 \cdot \pi_2 \cdot \pi_3$ is a positive v-cycle template with respect to n. The type-3 reachability criteria for fixed cycle templates $\ell = \pi_1 \cdot \pi_2 \cdot \pi_3$ in G and $\ell' = \pi'_1 \cdot \pi'_2 \cdot \pi'_3$ in G^{op} and a fixed s-t support F can now be expressed as follows, where $\varphi(G, \pi_1, \pi_2)$ and $\varphi(G, \pi'_1, \pi'_2)$ are the formulae from (4.6) and $\varphi(G, F, s, t)(c, c')$ is the formula from Lemma 4.1.7:

$$\varphi(G, F, s, t, \ell, \ell')(c, c') \stackrel{\text{def}}{=} \underbrace{\varphi(G, \pi_1, \pi_2)(c)}_{\text{suitable positive cycle at } s} \wedge \underbrace{\varphi(G^{\mathsf{op}}, \pi'_1, \pi'_2)(c')}_{\text{suitable positive cycle at } t \text{ in } G^{\mathsf{op}}} \wedge \\ \underbrace{\varphi(G, F, s, t)(c, c')}_{\text{suitable path flow with weight } c'-c} \quad (4.7)$$

Note that $\varphi(G, F, \ell, \ell')(c, c')$ is computable in polynomial time.

Lemma 4.1.16 *Given a graph G and vertices s, t the set*

$$\{(n, n') : \textit{ there exists an } s\text{-}t \textit{ path flow } f \textit{ such that } (G, f, n, n') \textit{ is a} \\ \textit{type-3 reachability certificate}\}$$

is definable via a set $R_3(G, s, t)$ of QFPA formulae, where $|\varphi| = O(|G|^2)$ for each $\varphi \in R_3(G, s, t)$.

Proof. For each positive s- and t-cycle template ℓ and ℓ' in G respectively G^{op} and for each s-t support F, R_3 is the smallest set containing a formula $\varphi(G, F, s, t, \ell, \ell')$ as defined in Equation (4.7). It is easily verified that each formula in $\varphi \in R_3$ has size $|\varphi| = O(|G|^2)$ and using Lemma 4.1.7 and our reasoning above it is easily checked that R_3 has the desired properties. □

Putting the pieces together, we can now show that the reachability set for zero-test free one-counter automata is definable via a set of QFPA formulae.

Lemma 4.1.17 *Let $\mathcal{A}$ be a zero-test free one-counter automaton, the set*

$$\{(n, n') : (q, n) \to^*_{\mathcal{A}} (q', n')\}$$

is definable via a set $R^z_{\mathcal{A}}(q, q')$ of QFPA formulae, where $|\varphi| = O(|G_{\mathcal{A}}|^4)$ for each $\varphi \in R_{\mathcal{A}}(q, q')$.

Proof. For any $q_1, q_2 \in Q$, $R^z_{\mathcal{A}}(q, q')$ is defined as the smallest set consisting of QFPA formulae open in c, c' of the form $\exists c_1, c_2.\varphi_1(c, c_1) \wedge \varphi_3(c_1, c_2) \wedge \varphi_2(c_2, c')$ where $\varphi_1 \in R_1(q, q_1)$, $\varphi_3 \in R_3(q_1, q_2)$ and $\varphi_2 \in R_2(q_2, q')$. Lemma 4.1.14 states that $(q, n) \to^*_{\mathcal{A}} (q', n')$ if, and only if, there is a path π that can be written as $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ and $n_1, n_2 \in \mathbb{N}$ such that in the most general case $(G_{\mathcal{A}}, f_{\pi_1}, n, n_1)$, $(G_{\mathcal{A}}, f_{\pi_2}, n_1, n_2)$ and $(G_{\mathcal{A}}, f_{\pi_3}, n_2, n')$ each yield type-1, type-3 respectively type-2 reachability certificates. Thus, assuming $(q, n) \to^*_{\mathcal{A}} (q', n')$, by Lemma 4.1.15 there exists $\varphi_1(c, c_1) \in R_1(G_{\mathcal{A}}, q, q_1)$ such that $\varphi_1[n/c, n_1/c_1]$. By the same lemma, there exists $\varphi_2(c_2, c') \in R_2(G_{\mathcal{A}}, q, q_1)$ such that $\varphi_2[n_2/c_2, n'/c']$. Finally, Lemma 4.1.16 yields that there exists $\varphi_3(c_1, c_2) \in R_3(G_{\mathcal{A}}, q_1, q_2)$ such that $\varphi_3[n_1/c_1, n_2/c_2]$. Hence, $\varphi[n/c, n'/c']$ for some $\varphi(c, c') \in R^z_{\mathcal{A}}(G_{\mathcal{A}}, q, q')$.

Conversely, if $\varphi[n/c, n'/c']$ for some $\varphi(c, c') \in R^z_{\mathcal{A}}(q, q')$ then by the Lemmas 4.1.15 and 4.1.16 the reachability certificates required in Lemma 4.1.14 exist. □

It is now an easy task to generalise this approach to one-counter automata with zero tests. The main idea is that any edge testing the counter for zero is traversed at most once on a run $\varrho : (q, n) \to^*_{\mathcal{A}} (q', n')$. Indeed, if ϱ can be factored as $\varrho = \varrho_1 \cdot (p, 0)(p', 0) \cdot \varrho_2 \cdot (p, 0)(p', 0) \cdot \varrho_3$ where $\xi(p, p') = \mathsf{zero}$ then clearly $\varrho' \stackrel{\text{def}}{=} \varrho_1 \cdot (p, 0)(p', 0) \cdot \varrho_3$ is a run in which the transition (p, p') is traversed one time less than in ϱ.

Lemma 4.1.18 *Let $\mathcal{A}$ be a one-counter automaton, the set*

$$\{(n, n') : (q, n) \to^*_{\mathcal{A}} (q', n')\}$$

is definable via a set $R_{\mathcal{A}}(q, q')$ of QFPA formulae, where $|\varphi| = O(|G_{\mathcal{A}}|^5)$ for each $\varphi \in R_{\mathcal{A}}(q, q')$.

Proof. We obtain from $\mathcal{A}$ zero-test free one-counter automata $\mathcal{A}_1, \ldots, \mathcal{A}_i$ with $i \in [|\mathcal{A}|]$ such that any run ϱ factors into $\varrho = \varrho_1 \cdot (q'_1, 0)(q_2, 0) \cdot \varrho_2 \cdot \ldots \cdot (q'_{i-1}, 0)(q_i, 0) \cdot \varrho_i$ where $\xi(q'_i, q_{i+1}) = \mathsf{zero}$ and each ϱ_i is a run only involving control locations from the zero-test free one-counter automaton $\mathcal{A}_i$. Hence, for any possible combination of zero-test free automata $\mathcal{A}_1, \ldots, \mathcal{A}_i$ and locations q_i, q'_i, $R_{\mathcal{A}}(q, q')$ is the smallest set containing a formula

$$\varphi(c, c') \stackrel{\text{def}}{=} \exists_{j \in [i]} c_j. \bigwedge_{j \in [i]} c_j = 0 \wedge \varphi_1(c, c_1) \wedge \varphi_j(c_j, c) \wedge \bigwedge_{j \in [2, i-1]} \varphi(c_j, c_{j+1}),$$

where $\varphi_1(c, c_1) \in R^z_{\mathcal{A}_1}(q, q'_1)$, $\varphi_i(c_i, c') \in R^z_{\mathcal{A}_i}(q_i, q')$ and $\varphi_j(c_j, c_{j+1}) \in R^z_{\mathcal{A}_j}(q_j, q'_j)$ for $j \in [2, i-1]$. Here, $R^z_{\mathcal{A}_j}$ are the sets of QFPA formulae from Lemma 4.1.17. It is easily verified that the statement of the lemma follows as a straightforward consequence from Lemma 4.1.17. □

The previous lemma now immediately gives us one of the main results of this chapter.

Theorem 4.1.1 *Reachability in one-counter automata is* NP*-complete.*

Proof. Let $\mathcal{A}$ be a one-counter automaton and (q, n) and (q', n') be configurations of $\mathcal{A}$. By Lemma 4.1.18, $(q, n) \to^*_{\mathcal{A}} (q', n')$ if, and only if, there exists $\varphi(c, c') \in R_{\mathcal{A}}(q, q')$ such that $\varphi[n/c, n'/c']$. An NP-algorithm that decides reachability non-deterministically guesses all components needed to compute such a formula $\varphi(c, c') \in R_{\mathcal{A}}(q, q')$, *i.e.*, the order in which zero-edges from $\mathcal{A}$ are traversed and the implied zero-test free automata, the support-edge decompositions for the type-1 and type-2 reachability certificates, and the positive cycle templates and support required for type-3 reachability certificates. It then computes $\varphi(c, c')$ in polynomial time, checks in NP satisfiability of $\exists c, c'.\varphi(c, c') \wedge c = n \wedge c' = n'$ and returns the result of the satisfiability check. □

We close this section with considering deciding the existence of Büchi paths in $T(\mathcal{A})$ and boundedness of one-counter counter automata. For both problems, we construct sets of QFPA formulae that characterise the set of configurations for which a Büchi path exist respectively for which the reachability set is unbounded. To this end, we incorporate the reachability sets $R_{\mathfrak{A}}(q, q')$ defined previously.

We begin with Büchi paths and provide sufficient and necessary criteria for the existence of Büchi paths. In the following, let F be the set of final locations of $\mathcal{A}$.

Lemma 4.1.19 *A one-counter automaton $\mathcal{A}$ has a Büchi path starting in (q, n) if, and only if, there are $q_f \in F$ and $n' \in \mathbb{N}$ such that*

(i) there are $n'' \in \mathbb{N}$ and runs ϱ_1, ϱ_2 such that $n'' \geq n$, $\varrho_1 : (q, n) \to_{\mathcal{A}}^* (q_f, n')$, $\varrho_2 : (q_f, n') \to_{\mathcal{A}}^* (q_f, n'')$ and ϱ_2 is zero-test free; or

(ii) there are runs ϱ_1, ϱ_3 such that $\varrho_1 : (q, n) \to_{\mathcal{A}}^* (q_f, n')$ and $\varrho_3 : (q_f, n') \to_{\mathcal{A}}^* (q_f, n')$.

Proof. ("$\Rightarrow$") Suppose $\mathcal{A}$ has a Büchi path starting in (q, n) and let ϱ be a run starting in (q, n) such that $q_f \in \mathit{inf}(\varrho) \cap F$. Let $(q_f, n_1), (q_f, n_2), \ldots$ be the q_f-configurations visited in ϱ. Our first observation is that for any $i \in \mathbb{N}_{>0}$, there is some $j > i$ such that $n_j \geq n_i$ since there are no infinite descending chains in $\mathbb{N}$. If there is a zero-test free segment $(q_f, n_i) \to_{\mathcal{A}}^* (q_f, n_j)$ with $j > i$ and $n_j \geq n_i$ in ϱ then the conditions in (i) are fulfilled by setting $n' = n_i$ and $n'' = n_j$. Otherwise, some transition testing the counter value for zero is visited infinitely often. Hence there is some $(q', 0)$ occurring infinitely often in ϱ for some $q' \in Q$. Hence ϱ has a prefix $\varrho_1' : (q, n) \to_{\mathcal{A}}^* (q', 0)$. Moreover, there are segments $\varrho_1'' : (q', 0) \to_{\mathcal{A}}^* (q_f, n_i)$ and $\varrho_3' : (q_f, n_i) \to_{\mathcal{A}}^* (q', 0)$ since q_f and $(q', 0)$ occur infinitely often in ϱ for some $i \in \mathbb{N}_{>0}$. By setting $n' = n_i$, $\varrho_1 = \varrho_1' \cdot \varrho_1''$ and $\varrho_3 = \varrho_1'' \cdot \varrho_3'$ we obtain the runs $\varrho_1 : (q, n) \to_{\mathcal{A}}^* (q_f, n')$ and $\varrho_3 : (q_f, n') \to_{\mathcal{A}}^* (q_f, n')$ as required in condition (ii).

("$\Leftarrow$") We define an infinite run ϱ on which q_f occurs infinitely often. In case (i), since ϱ_2 is zero-test free, we have $(q_f, n' + d) \to_{\mathcal{A}}^* (q_f, n'' + d)$ for any $d \geq 0$. Hence, we can define the required run to be $\varrho : \varrho_1 \cdot \varrho_2 \cdot (\varrho_2 + n'' - n') \cdot (\varrho_2 + 2(n'' - n')) \ldots$. In case (ii), we obviously have that $\varrho : \varrho_1 \cdot \varrho_3 \cdot \varrho_3 \cdot \varrho_3 \ldots$ is a suitable run in $T(\mathcal{A})$. □

It is now clear that we can translate the conditions from the previous lemma into a sentence in QFPA, which gives us NP membership of checking for the existence of a Büchi path. Hardness for NP easily follows from an adoption of Proposition 4.1.2.

Lemma 4.1.20 *Let $\mathcal{A}$ be a one-counter automaton and $q \in Q$, the set*

$$\{n : \text{there is a Büchi path starting at } (q, n) \text{ in } T(\mathcal{A})\}$$

is definable via a set $R^B_{\mathcal{A}}(q)$ of QFPA formulae, where $|\varphi| = O(|G_{\mathcal{A}}|^5)$ for each $\varphi \in R^B_{\mathcal{A}}(q)$.

Proof. We encode the conditions from Lemma 4.1.19 into a sentence in QFPA. Let $\mathcal{A}'$ be obtained from $\mathcal{A}$ by removing all edges labelled with zero. The set $R^B_{\mathcal{A}}(q)$ is the smallest set containing for each pair $q_f \in F$ and $q \in Q$ a formula $\varphi(c)$ such that

$$\varphi(c) \overset{\text{def}}{=} \exists c'. \left((\exists c''.c'' \geq c' \wedge \varphi_1(c, c') \wedge \varphi_2(c', c'')) \vee (\varphi_3(c, c') \wedge \varphi_4(c', c))\right),$$

where $\varphi_1(c, c') \in R_{\mathcal{A}}(q, q_f)$, $\varphi_2(c', c'') \in R_{\mathcal{A}'}(q_f, q_f)$, $\varphi_3(c, c') \in R_{\mathcal{A}}(q, q_f)$ and $\varphi_4(c', c') \in R_{\mathcal{A}}(q_f, q_f)$. The correctness of the lemma is an immediate consequence of the lemmas 4.1.18 and 4.1.19. □

Theorem 4.1.2 *Deciding the existence of a Büchi path for one-counter automata is* NP*-complete.*

Finally, we provide a solution to the problem left open by Rossier and Yen in [95] and show that boundedness for one-counter automata is NP-complete. Recall that for a given one-counter automaton $\mathcal{A}$ and a configuration (q, n), boundedness is to decide whether the set

$$\{(q', n') \in Q \times \mathbb{N} : (q, n) \to^*_{\mathcal{A}} (q', n')\}$$

is infinite. As observed in [95], deciding boundedness boils down to checking if we can reach a configuration from which we can loop with a strictly positive counter increment. We define the set $B_{\mathcal{A}}(q)$ of QFPA formulae to be the smallest set containing for each $q' \in Q$ a QFPA formula

$$\varphi(c) \overset{\text{def}}{=} \exists c', c''. \varphi_{q,q'}(c, c') \wedge \varphi_{q',q'}(c', c'') \wedge c'' > c',$$

where $\varphi_{q,q'}(c, c') \in R_{\mathcal{A}}(q, q')$ and $\varphi_{q',q'} \in R_{\mathcal{A}}(q', q')$. Clearly, for any $n \in \mathbb{N}$ and $\varphi(c) \in B_{\mathcal{A}}(q)$, $\varphi[c/n]$ is satisfiable if, and only if, the reachability set at (q, n) is infinite. Moreover, $|\varphi| = O(|\mathcal{A}|^5)$ for each $\varphi \in B_{\mathcal{A}}(q)$.

Theorem 4.1.3 *Deciding boundedness for one-counter is* NP*-complete.*

4.2 Reachability in Parametric Counter Automata

In this section, we are going to establish the complexity of reachability in parametric counter automata. As discussed in Chapter 2, reachability in k-counter automata with $k \geq 2$ is undecidable, but decidable for any k if we consider zero-test-free k-counter automata. The first result of this section is that even if we restrict ourselves to zero-test-free parametric k-counter automata, the reachability problem is undecidable for $k \geq 4$.

Theorem 4.2.1 *The reachability problem for zero-test-free parametric k-counter automata is Σ^0_1-complete for $k \geq 4$.*

Proof. Regarding hardness, we reduce from the reachability problem for two-counter automata. Given a two-counter automaton $\mathcal{A}$, we derive from $\mathcal{A}$ a zero-test-free parametric four-counter automaton $\mathcal{A}'$ with one parameter y such that for any two control locations q, q' of $\mathcal{A}$ we have $(q, 0) \rightarrow^*_{\mathcal{A}} (q', 0)$ if, and only if, $(q_0, \vec{0}) \rightarrow^*_{\mathcal{A}'} (q'_0, \vec{0})$ for some designated control locations q_0, q'_0 of $\mathcal{A}'$.

Suppose that there is a run ϱ starting in $(q, 0)$ and ending in $(q', 0)$ in $T(\mathcal{A})$. Since r is finite, each counter of $\mathcal{A}$ does not grow above some $m \in \mathbb{N}$. Our aim is to use the parameter y in order to guess this maximum value m so that we can simulate a run of $\mathcal{A}$ by $\mathcal{A}'$. We adopt an idea introduced by Lipton [76]. During an emulation of a run of $\mathcal{A}$, the first counter of $\mathcal{A}'$ stores the value n_1 of the first counter and the second counter of $\mathcal{A}'$ stores $m - n_1$, ensuring as an invariant that the sum of the value of the first and the second counter is m. Likewise, the third counter of $\mathcal{A}'$ stores the value n_2 of the second counter of $\mathcal{A}$ and the fourth counter of $\mathcal{A}'$ stores $m - n_2$. Performing a zero-test on the first respectively second counter of $\mathcal{A}$ can then be simulated by adding and subtracting y from the second respectively fourth counter of $\mathcal{A}'$. Thus, for any instantiation of y, provided that the above invariants hold, $\mathcal{A}'$ can correctly simulate all runs of $\mathcal{A}$ in which the counter value does not exceed the value of y. In order to construct $\mathcal{A}'$ from $\mathcal{A}$ we introduce an extra control location in between all transitions of $\mathcal{A}$, as shown by the replacement rules in Figure 4.3: any $\mathsf{add}_i(z)$-operation on the i-th counter of $\mathcal{A}$ is replaced by two consecutive

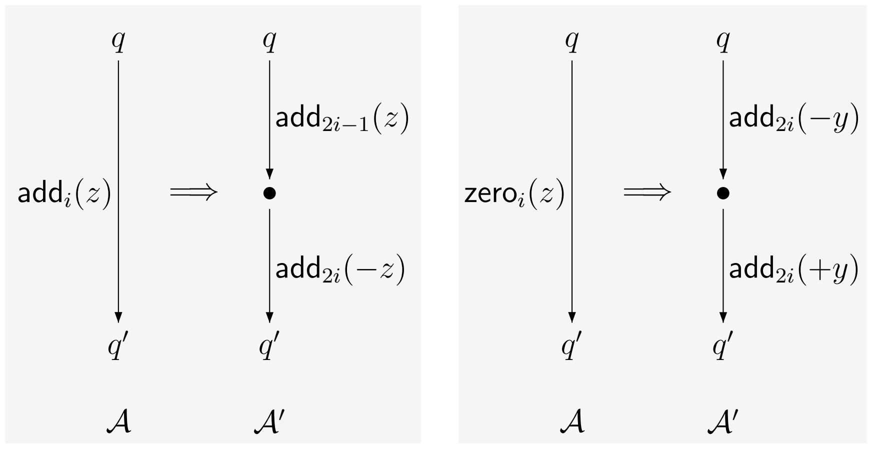

Figure 4.3: Replacement rules for the reduction from reachability in a two-counter automaton $\mathcal{A}$ to reachability in zero-test free parametric four-counter automaton $\mathcal{A}'$. Here $\mathcal{A}'$ is obtained by replacing each transition on the left-hand side of the $\Longrightarrow$-arrow by the transitions on the right-hand side of $\Longrightarrow$ arrow.

add-operations on the $(2i-1)$-th and the $2i$-th counter of $\mathcal{A}'$, and any zero_i-operation on counter i is replaced by consecutively subtracting and adding y to the counter $2i-1$. The only missing piece is the initialisation part of $\mathcal{A}'$ that initially establishes the invariant between the counters of $\mathcal{A}'$. But this is trivially done by introducing a new control location q_0 and by adding y to the second and fourth counter along a unique path connecting q_0 to q. Likewise, we connect q' to a new location q_0' and along the transition from q' to q_0' we subtract y from the second and the fourth counter. Clearly, $(q,0) \to_{\mathcal{A}}^* (q',0)$ if, and only if, $(q_0,\vec{0}) \to_{\mathcal{A}'}^* (q_0',\vec{0})$.

Regarding membership in Σ_1^0, we can enumerate all possible valuations ν of the parameters and check whether $(q,\vec{n}) \to_{\mathcal{A}}^* (q',\vec{n'})$, which is decidable by Theorem 4.1.1. $\square$

In contrast to this negative result, we show in the remainder of this section that reachability is decidable for parametric *one*-counter automata with zero tests. This result is shown by generalising the concepts and techniques from Section 4.1. Reachability relations in parametric one-counter automata are not definable in QFPA, but, as we are going to show below, definable in QFPAD. Deciding reachability and related problems for various classes of counter machines via a reduction to QFPAD is a tool that has extensively been used in the literature, see *e.g.* [64, 38, 63].

Before we begin, we provide a reduction in the converse direction and show that satisfiability in QFPAD is reducible to reachability in parametric one-counter automata. This reduction serves two purposes. First, it is an interesting fact that satisfiability in QFPAD can be rephrased in automata-theoretic terms. It is known [21] that satisfiability in QFPA is reducible to emptiness in non-deterministic finite-state automata, and thus we produce here a result in a similar spirit for QFPAD. Second, the reduction provided strengthens the NP-hardness result for reachability. Recall that satisfiability in QFPAD is already NP-complete for a QFPAD formula with a *fixed* number of Boolean connectives. The subsequent result shows that shows that reachability in parametric one-counter automata is already NP-hard for a fixed number of control locations.

Lemma 4.2.1 *Let $\varphi(\vec{y})$ be a QFPAD formula. There exists a parametric one-counter automaton $\mathcal{A}$ with control locations q, q' such that φ is satisfiable if, and only if, $(q, 0) \rightarrow^*_{\mathcal{A}} (q', 0)$.*

Proof. Let $\vec{y} = (y_1, \ldots, y_n)$, for our purposes we may assume with no loss of generality that $y_1, \ldots, y_n$ are all variables occurring in φ, *i.e.*, no $\exists$-quantifier occurs in φ. Moreover, we assume that no negation symbol occurs in φ. Any QFPAD formula can be transformed into negation normal form and by applying the procedure described in Section 2.6.2, *i.e.*, by introducing additional first-order slack variables, non-divisibilities in φ can be eliminated. Furthermore, we may assume that coefficients in φ are encoded in unary. Moreover, we subsequently assume that the first-order variables in *any* linear polynomial p from φ are ordered in a way such that variables with positive coefficient are written first and variable with negative coefficients last, *i.e.*, any p is written as $p = y_{i,1} + \ldots + y_{i,j} + z - y_{i,j+1} - \ldots - y_{i,m}$, where $z \in \mathbb{Z}$.

Figure 4.4 shows the gadgets that are the building blocks of our reduction. On top, the parametric one-counter automaton $\mathcal{A}_p$ for a single linear polynomial $p(\vec{y})$ is

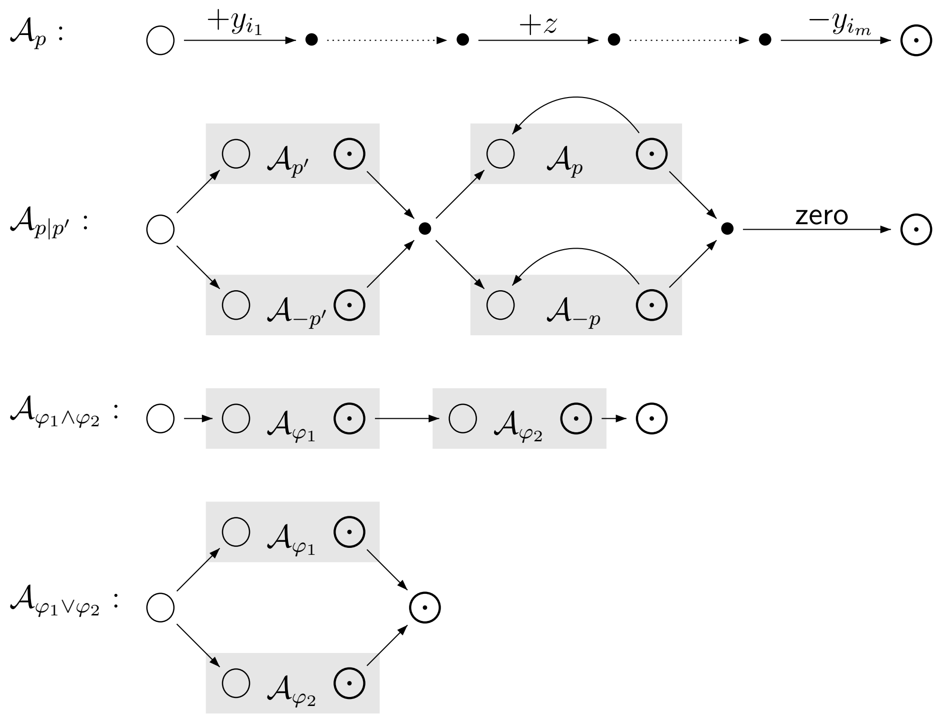
$\mathcal{A}_p$:
$+y_{i_1}$
$+z$
$-y_{i_m}$
$\mathcal{A}_{p|p'}$:
$\mathcal{A}_{p'}$
$\mathcal{A}_{-p'}$
$\mathcal{A}_p$
$\mathcal{A}_{-p}$
zero
$\mathcal{A}_{\varphi_1 \wedge \varphi_2}$:
$\mathcal{A}_{\varphi_1}$
$\mathcal{A}_{\varphi_2}$
$\mathcal{A}_{\varphi_1 \vee \varphi_2}$:
$\mathcal{A}_{\varphi_1}$
$\mathcal{A}_{\varphi_2}$

Figure 4.4: Gadgets of the parametric one-counter automaton constructed in the reduction from QFPAD to a reachability instance.

given. One row below is the automaton $\mathcal{A}_{p|p'}$ used to handle divisibilities $p|p'$ from φ. The automaton $\mathcal{A}_{p|p'}$ first non-deterministically chooses to add the valuation of the linear polynomial p' or $-p'$ to the counter. It then repeatedly subtracts either p or $-p$ from the counter until the counter value is 0. Since under any valuation we have $p|p' \Leftrightarrow p| - p' \Leftrightarrow -p| - p' \Leftrightarrow -p|p'$ it follows that under any valuation, whenever we can reach the location $\odot$ from $\bigcirc$ then the value of p divides the value of p'. Here, it is important that the parameters in the linear polynomials are ordered according to their sign, roughly speaking, in order to prevent the automaton from getting stuck. Boolean connectives are handled in a straightforward fashion, *i.e.*, the automaton $\mathcal{A}_{\varphi_1 \wedge \varphi_2}$ first runs through the automaton that corresponds to φ_1 and then through the automaton corresponds to φ_2. Likewise, disjunction is handled by $\mathcal{A}_{\varphi_1 \vee \varphi_2}$ via branching. It is now clear how to define the automaton $\mathcal{A}_\varphi$ by structural induction for any φ and that $\mathcal{A}_\varphi$ has the desired properties. $\square$

Corollary 4.2.1 *Reachability in parametric one-counter automata is* NP*-hard already for a fixed number of control locations.*

We now show that for any reachability problem in a parametric one-counter automaton we can construct a QFPAD formula φ such that reachability holds if, and only if, φ is satisfiable. To this end, we generalise the approach taken in Section 4.1. For the remainder of this section, let us fix a parametric one-counter automaton $\mathcal{A} = (Q, Y, \Lambda, q_0, F, \Delta, \lambda, \xi)$ with $Y = \{y_1, \ldots, y_k\}$. Given an instance $(q, n), (q', n')$ of a reachability problem and let $\vec{y} = (y_1, \ldots, y_k)$, we are going to show that the set

$$\{(\nu(\vec{y}), n, n') \in \mathbb{N}^{k+2} : \nu(\vec{y}) = (m_1, \ldots, m_k), (q, n) \rightarrow^*_{\mathcal{A}^\nu} (q', n')\}$$

is QFPAD-definable. Here, we have lifted valuations to vectors, *i.e.*, $\nu(\vec{y}) \overset{\text{def}}{=} (\nu(y_1), \ldots, \nu(y_k))$.

As a first step, we generalise weighted graphs to *parametric weighted graphs*, which we sometimes just call *parametric graphs*. Similar to a weighted graph, a parametric graph is a tuple $G = (V, Y, E, \mu)$, where μ can additionally label a transition with a parameter from Y, *i.e.*, $\mu : E \to \mathbb{Z} \cup \{\circ y : \circ \in \{+, -\}, y \in Y\}$. Given a valuation of the parameters $\nu : Y \to \mathbb{N}$, we denote by G^ν the weighted graph obtained from replacing every label $\circ y \in Y$ of G by $\circ\nu(y)$. All other definitions from weighted graphs carry over straightforwardly and parameters are treated symbolically. For example, given a path flow $f : E \to \mathbb{N}$, the weight of f becomes a linear polynomial in $\vec{y}$ instead of an integer:

$$\textit{weight}(G, f)(\vec{y}) \overset{\text{def}}{=} \sum_{e \in E} f(e)\mu(e).$$

Recall that for a weighted graph G and an s-t support F, in Lemma 4.1.1 we gave a QFPA formula $\varphi(G, F, s, t)(c, c')$ such that for all $n, n \in \mathbb{N}$, $\varphi(G, F, s, t)[n/c, n'/c']$ is satisfiable if, and only if, there exists an s-t path flow f with support F such that $\textit{weight}(G, f) = n' - n$. For a parametric graph G, the analogous question is to decide whether there exists a valuation of the parameters such that there is an s-t path flow with a certain weight. This problem is expressible in the existential theory $\langle \mathbb{N}, <, +, \cdot, 0, 1 \rangle$, where as in Section 4.1 we treat the weights $f(e)$ of f as first-order variables:

$$\varphi(G, F, \vec{y}, s, t)(c, c') \overset{\text{def}}{=} \exists y_1, \ldots, y_k . \exists_{e \in E} f(e) . \psi(G, s, t) \wedge \\ \wedge \sum_{e \in F} f(e)\mu(e) = c' - c \wedge \bigwedge_{e \in E \setminus F} f(e) = 0.$$

Here, $\psi(G, s, t)$ is the QFPA formula open in $f(e), e \in E$ corresponding to the Eulerian path flow conditions from Lemma 4.1.6. The drawback is of course that the existential theory of $\langle \mathbb{N}, <, +, \cdot, 0, 1\rangle$ is undecidable. However, using the decomposition of path flows as loops and simple paths given in Section 4.1.3, we obtain a set of equi-satisfiable QFPAD formulae. We thus now consider those two types of graphs separately.

Let G be a graph with parameters $\vec{y} = (y_1, \ldots, y_k)$, let F be a v-v support such that G/F is a loop and let $n, n' \in \mathbb{N}$. By the Eulerian path flow conditions given in Lemma 4.1.6, any path flow f assigns the *same* weight to each of the edges in F. Consequently, if we are asking for the existence of a valuation of the parameters and a v-v path flow f with support F and a weight $z = n' - n$, the valuation and the path flow exist if the total weight of the loop G/F divides z. This expresses in QFPAD as follows, where the last conjunct ensures that loops with negative weight can only contribute to a total negative weight and *vice versa*:

$$\varphi_\ell(G, F)(\vec{y}, c, c') \overset{\text{def}}{=} \sum_{e \in F} \mu(e) | c' - c \wedge \sum_{e \in F} \mu(e) > 0 \leftrightarrow c' - c > 0. \tag{4.8}$$

It follows that for any valuation ν, $\varphi_\ell(G, F)[\nu(\vec{y})/\vec{y}, n/c, n'/c']$ holds if, and only if, there exists a v-v path flow f in G^ν such that $\mathit{weight}(G^\nu, f) = n' - n$.

Next, we consider the case when G/F is a simple s-t path. This case is trivial, since any s-t path flow can only assign weight one to each edge in F. We set

$$\varphi_p(G, F)(\vec{y}, c, c') \overset{\text{def}}{=} \sum_{e \in F} \mu(e) = c' - c. \tag{4.9}$$

Clearly for any valuation ν, $\varphi(G, F)[\nu(\vec{y})/\vec{y}, n/c, n'/c']$ holds if, and only if, there exists an s-t path flow f such that $\mathit{weight}(G^\nu, f) = n' - n$.

We can now use the decomposition provided in Lemma 4.1.10 in order to show that the existence of path flows in parametric graphs is definable in QFPAD.

Lemma 4.2.2 *Given a parametric graph G, vertices s, t and an s-t support F, the set*

$$\{(\nu(\vec{y}), n, n') : \textit{there exists an } s\text{-}t \textit{ path flow } f \textit{ with support } F \quad \textit{such that } \textit{weight}(G^\nu, f) = n' - n\}$$

is definable via a set $P(G, F, s, t)$ of QFPAD formulae, where $|\varphi| \in O(|G|^3)$ for each $\varphi \in P(G, s, t)$.

Proof. For any sequence of supports $\vec{F} = (F_0, F_1, \ldots, F_j)$, $j \in [|G|^2]$ such that G/F_0 is a simple s-t path, G/F_i is a simple loop and $F = \bigcup_{i \in [0,j]} F_i$, $P(G, F, s, t)$ is the smallest set containing a QFPAD formula

$$\varphi(G, \vec{F})(\vec{y}, c, c') \overset{\text{def}}{=} \exists_{i \in [0,j]} c_i, c_i'.\varphi_p(G, F_0)(\vec{y}, c_0, c_0') \wedge \bigwedge_{i \in [j]} \varphi_\ell(G, F_i)(\vec{y}, c_i, c_i') \wedge \wedge \sum_{i \in [0,j]} c_i' - c_i = c' - c,$$

where φ_p and φ_ℓ are defined as in Equation (4.9) respectively (4.8). Both $|\varphi_p| = O(|G|)$ and $|\varphi_\ell| = O(|G|)$, hence $|\varphi(G, F, s, t)| = O(|G|^3)$.

Suppose $\varphi(G, \vec{F})[\nu(\vec{y})/\vec{y}, n/c, n'/c']$ holds for some $\varphi(G, \vec{F}) \in P(G, s, t)$, a valuation ν and $n, n' \in \mathbb{N}$. Applying the semantics of φ_p and φ_ℓ, we conclude that there are path flows $f_0, f_1, \ldots, f_j$ and $n_i, n_i' \in \mathbb{N}$ such that $\textit{weight}(G^\nu, f_i) = n_i' - n_i$ and $\sum_{i \in [0,j]} \textit{weight}(G^\nu, f_i) = n' - n$. It follows from Lemma 4.1.8 that $f \overset{\text{def}}{=} \sum_{[0,j]} f_i$ is a path flow. Moreover, $\textit{weight}(G^\nu, f) = n' - n$ as required.

Conversely, assume that f is an s-t path flow such that $\textit{weight}(G^\nu, f) = n' - n$ for some valuation ν and $n, n' \in \mathbb{N}$. By Lemma 4.1.10 there exist path flows $f_0, f_1, \ldots, f_j, j \in [|G|^2]$ such that G/F_0 is a simple s-t path and each G/F_i is a loop for $i \in [j]$. By construction of $P(G, s, t)$, there exists some $\varphi(G, \vec{F})$ that is satisfiable for this particular decomposition. □

This lemma is basically all that is needed in order to show that reachability in parametric one-counter automata is in NP. Everything else is just a straightforward adaption of the proof given for one-counter automata without parameters, since through the parameters, most formulae constructed in Section 4.1 just become open formulae in those parameters. The lemmas 4.1.15 and 4.1.16 which construct sets of QFPA formulae that guarantee the existence of type-1, type-2 and type-3 reachability certificates need to be adjusted to cater our needs. We defer details to the end of this section and, for now, assume that there exist sets $R_1(G, s, t), R_2(G, s, t)$ and $R_3(G, s, t)$ that define the sets

$$\begin{aligned} M_i(G, s, t) \overset{\text{def}}{=} \{(\nu(\vec{y}), n, n') \in \mathbb{N}^{k+2} : \text{there exists an } s\text{-}t \text{ path flow } f \text{ such that} \\ (G^\nu, f, n, n') \text{ is a type-}i \text{ reachability certificate}\} \end{aligned}$$

for each $i \in [3]$. In order to show that reachability in parametric one-counter automata is in NP, instead of repeating the proof given in Section 4.1, we now only sketch differences and slight adjustments to the key lemmas in Section 4.1.

- The definition of the formulae given in Lemma 4.1.1 which express in QFPA the weight and the drop of a path can be reused in order to obtain QFPA formulae $\varphi_w(G, \pi)(\vec{y}, c, c')$ and $\varphi_d(G, \pi)(\vec{y}, c, c')$ such that for any path π,

$$\begin{aligned} \varphi_w(G, \pi)[\nu(\vec{y})/\vec{y}, n/c, n'/c'] &\Leftrightarrow \textit{weight}(G^\nu, \pi) = n' - n; \text{ and} \\ \varphi_d(G, \pi)[\nu(\vec{y})/\vec{y}, n/c, n'/c'] &\Leftrightarrow \textit{drop}(G^\nu, \pi) = n' - n. \end{aligned}$$

- Building upon the generalization of φ_w and φ_d, we obtain an analogue to the QFPA formula defined in Equation (4.6) that allows for determining if a path π that can be factored as $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ yields a positive cycle template with respect to some $n \in \mathbb{N}$. More specifically, we can construct a QFPA formula $\varphi(G, \pi_1, \pi_2)(\vec{y}, c)$ such that $\varphi(G, \pi_1, \pi_2)[\nu(\vec{y})/\vec{y}, n/c]$ holds if, and only if, π is a positive cycle template in G^ν with respect to n.

- The encoding of the Bellman-Ford algorithm in QFPA in Lemma 4.1.5 can directly be used for parametric graphs such that we obtain a formula $\varphi(G)(\vec{y})$ such that $\varphi(G)[\nu(\vec{y})/\vec{y}]$ holds if, and only if, G^ν does not contain any positive cycle.

- Assuming we have shown the analogue statements to the lemmas 4.1.15 and 4.1.16 and thus have sets $R_i(G, s, t)$ as described above, we can easily adopt the proof of Lemma 4.1.17 in order to obtain a set $R^z_{\mathcal{A}}(q, q')$ of QFPAD formulae that defines the set

  $$\{(\nu(\vec{y}), n, n') \in \mathbb{N}^{k+2} : (q, n) \to^*_{\mathcal{A}^\nu} (q', n')\}$$

  for zero-test free parametric one-counter automata.

- Finally, by guessing the order in which zero-test transitions are traversed in $\mathcal{A}$ and using the sets $R^z_{\mathcal{A}}(q, q')$ constructed above, we get the analogue to Lemma 4.1.18 and can construct for an arbitrary parametric one-counter automaton $\mathcal{A}$ and control locations q, q' a set $R_{\mathcal{A}}(q, q')$ of QFPAD formulae that defines the set

  $$\{(\nu(\vec{y}), n, n') \in \mathbb{N}^{k+2} : (q, n) \to^*_{\mathcal{A}^\nu} (q', n')\}.$$

  Moreover, since the size of the constructed formulae do not change in our generalisation, we have $|\varphi| = O(|G_{\mathcal{A}}|^5)$ for all $\varphi \in R_{\mathcal{A}}(q, q')$.

Since by Theorem 2.6.3 satisfiability in QFPAD is NP-complete, we obtain the main result of this section.

Theorem 4.2.2 *Reachability in parametric one-counter automata is* NP*-complete.*

As in the case of one-counter automata without parameters, this result allows us to conclude that checking the existence of a Büchi path in parametric one-counter automata is NP-complete as well via a straightforward adoption of the proof of Lemma 4.1.20.

Theorem 4.2.3 *Deciding the existence of a Büchi path in parametric one-counter automata is* NP*-complete.*

We close this section with the deferred construction of the sets $R_1(G, s, t)$, $R_2(G, s, t)$ and $R_3(G, s, t)$ for a parametric graph G and vertices s and t. The construction is analogue to the construction in Section 4.1. The main difference is that we have to incorporate Lemma 4.2.2.

Regarding R_1, suppose we are given G and vertices s, t. Let us fix an s-t support F and a support-edge decomposition $(F_i, v_i, v_i', e_i)_{i \in [m]}$. For each F_i, by Lemma 4.2.2 there exists a set $P(G, F_i, v_i, v_i')$ of QFPAD formulae that defines the set of valuations for which a v_i-v_i' path flow f_i exists. Each QFPAD formula in such a set $P(G, F_i, v_i, v_i')$ is determined by supports $F_{i,0}, F_{i,1}, \ldots, F_{i,j_i}$ for some $j_i \in [|G|^2]$ which give the decomposition of F_i in simple paths and cycles. Thus, for any fixed s-t support F, any fixed support-edge decomposition $(F_i, v_i, v_i', e_i)_{i \in [m]}$ of F and any fixed decomposition of all F_i into a simple path and loops $\vec{F_i} = (F_{i,0}, F_{i,1}, \ldots F_{i,i_j})$, $R_1(G, s, t)$ is the smallest set containing a QFPAD formula

$$\begin{aligned}\varphi \overset{\text{def}}{=} \exists_{i \in [m]} c_i, c_i'.\ & \underbrace{\varphi(G/F)(\vec{y})}_{\text{no positive cycles}} \wedge \\ \wedge \quad & \underbrace{\bigwedge_{i \in [m]} \varphi(G, \vec{F_i})(\vec{y}, c_i, c_i')}_{\text{there are path flows } f_i \text{ with weight } c_i' - c_i \text{ for the fixed decompositions } F_{i,0}, \ldots, F_{i,j_i}} \wedge \\ \wedge \quad & \underbrace{\bigwedge_{i \in [m]} \sum_{j \in [i]} c_i' - c_i + \mu(e_i) \geq -c}_{\text{weights of the edge decomposition sum up correctly}} \wedge \underbrace{\sum_{i \in [m]} c_i' - c_i + \mu(e_i) = c' - c}_{\text{total weight matches}}.\end{aligned}$$

Here, $\varphi(G/F)(\vec{y})$ is the QFPAD formula ensuring that no positive cycles exist and $\varphi(G, \vec{F_i})(\vec{y}, c_i, c_i')$ is the QFPAD formula from Lemma 4.2.2 for the fixed decomposition of each F_i. Since the size of $\varphi(G/F)(\vec{y})$ dominates the size of all other conjuncts, we have $|\varphi| = O(|G|^4)$ for each $\varphi \in R_1(G, s, t)$.

As in Section 4.1, the set R_2 can be defined as $R_2(G, s, t) \overset{\text{def}}{=} R_1(G^{\mathsf{op}}, t, s)$.

Finally, we are going to sketch the construction of the set $R_3(G, s, t)$. To this end, we fix a support F and cycle templates $\ell = \pi_1 \cdot \pi_2 \cdot \pi_3$ and $\ell' = \pi_1' \cdot \pi_2' \cdot \pi_3'$. As in the case of R_1, there is no single formula QFPAD that expresses the existence of a valuation ν and a path flow f with support F, and for that reason R_3 also needs to contain a formula for each formula in $P(G, F, s, t)$. Thus, for all fixed s-t supports F, all s-cycles ℓ that decompose as $\ell = \pi_1 \cdot \pi_2 \cdot \pi_3$, all t-cycles ℓ' that decompose as $\ell' = \pi_1' \cdot \pi_2' \cdot \pi_3'$ and all decompositions of F into a single path $\vec{F} = (F_0, F_1, \ldots, F_j)$

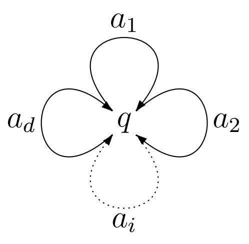

Figure 4.5: A simple bounded one-counter automaton.

for some $j \in [|G|^2]$, $R_3(G, s, t)$ is the smallest set containing a QFPAD formula

$$\varphi(G, F, s, t, \ell, \ell')(\vec{y}, c, c') \overset{\text{def}}{=} \underbrace{\varphi(G, \pi_1, \pi_2)(\vec{y}, c)}_{\text{suitable positive cycle at } s} \wedge \underbrace{\varphi(G^{\mathsf{op}}, \pi_1', \pi_2')(\vec{y}, c')}_{\text{suitable positive cycle at } t \text{ in } G^{\mathsf{op}}} \wedge$$
$$\underbrace{\varphi(G, \vec{F})(\vec{y}, c, c')}_{\text{there is an } s\text{-}t \text{ path flow for the fixed decomposition } F_0,\ldots,F_j \text{ of } F}$$

4.3 Bounded One-Counter Automata

The precise complexity of reachability in bounded one-counter automata remains an open problem of this thesis. Recall that bounded one-counter automata are one-counter automata in which the counter has to stay between zero and some upper bound $b \in \mathbb{N}$ on every run. At first sight, it might seem surprising that adding an additional constraint makes determining the complexity of reachability harder. However, a crucial observation in Section 4.1 was that we can use type-3 reachability certificates in order to pump up and down the counter as necessary. It is obvious that this approach does not work when there is an *a priori* upper bound on the counter.

In the remainder of this section, we are going to investigate aspects of reachability in a simple class of bounded one-counter automata for which in the general case we are unable to determine the precise complexity of reachability. However, we provide some indications that the problem might still be in NP. We call the class we are dealing with *simple bounded one-counter automata*. A simple bounded one-counter automaton is presented in Figure 4.5. For simplicity, we allow for multiple edges between control locations. Thus, a simple bounded one-counter automaton consists of one control location q and d self loops at q labelled with $a_i \in \mathbb{Z}$ and a bound $b \in \mathbb{N}$. It follows from Proposition 4.1.1 that the reachability problem for simple bounded one-counter automata is NP-hard. However, even if we are given the path flow of a run, there is no apparent upper bound below PSPACE to verify the existence of a run. Simple bounded one-counter automata thus provide the simplest instance of a class of bounded one-counter automata for which we cannot determine the precise complexity of reachability. The contribution of this section is that for $d = 2$ and a given path flow, we are subsequently going to show that it is possible to check for the existence of a run in polynomial time without actually constructing it as it is of potentially exponential length.

Our main idea is to associate with every reachability instance and a given path flow a polyhedron in a d-dimensional space. Checking reachability then reduces to checking for the existence of a lattice path to some designated point inside this polyhedron that corresponds to the path flow. For the special case $d = 2$, the existence of a lattice path can be related to the number of integral points inside the polyhedron, which is computable in polynomial time for the polyhedron that we obtain.

Definition 15 Given a simple bounded one-counter automaton $\mathcal{A}$ as depicted in Figure 4.5 with edges labelled with $\vec{a} = (a_1, \ldots, a_d) \in \mathbb{Z}^d$, a bound $b \in \mathbb{N}$ and $\vec{c} \in \mathbb{N}^d$, the *polyhedron $\mathcal{P}_{\mathcal{A}}(\vec{c}) \subseteq \mathbb{R}^d$ corresponding to $\mathcal{A}$* is defined as

$$\mathcal{P}_{\mathcal{A}}(\vec{c}) \overset{\text{def}}{=} \{\vec{x} \in \mathbb{R}^d : 0 \leq \vec{a} \cdot \vec{x} \leq b, 0 \leq x_i \leq c_i, i \in [d]\},$$

where $\vec{x} \cdot \vec{a}$ is the inner product of $\vec{x}$ and $\vec{a}$, and x_i and c_i denote the i-th component of $\vec{x}$ respectively $\vec{c}$. $\Diamond$

The grey shaded area in Figure 4.6 shows an example of a polyhedron $\mathcal{P}_{\mathcal{A}}(\vec{c})$ corresponding to a simple bounded one-counter automaton with two transitions such that $\vec{a} = (-5, 7)$, with bound $b = 11$ and $\vec{c} = (5, 4)$. The vector $\vec{c}$ can be thought of as a path flow and, as we are going to show below, reachability in $\mathcal{A}$ reduces to asking for the existence of a lattice path in $\mathcal{P}$ that starts in $\vec{0}$ and ends in $\vec{c}$. Formally, let $d \in \mathbb{N}_{>0}$ and for $i \in [d]$ let $\vec{u_i} \in \{0, 1\}^d$ denote the i-th unit vector, *i.e.*, the i-th component of $\vec{u_i}$ is equal to one and all other components are equal to zero.

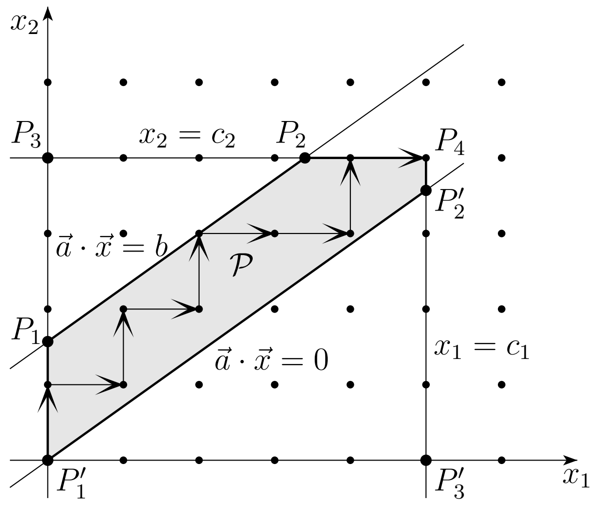

Figure 4.6: Example of a polyhedron $\mathcal{P}_\mathcal{A}(\vec{c})$ and a lattice path from $\vec{0}$ to $\vec{c}$, where $\vec{c} = (5,4)$ and $\vec{a} = (-5,7)$ and $b = 11$.

Definition 16 Let $\mathcal{P} \subseteq \mathbb{R}^d$ be a convex polyhedron and $\vec{c} \in \mathbb{N}^d$, a *lattice path ρ of length m starting in $\vec{0}$ and ending in $\vec{c}$ in $\mathcal{P}$* is a finite sequence of unit vectors $\vec{u_{i,1}} \dots \vec{u_{i,m}}$ such that $\sum_{k\in[j]} \vec{u_{i,k}} \in \mathcal{P}$ for all $j \in [0,m]$ and $\sum_{k\in[m]} \vec{u_{i,k}} = \vec{c}$. $\Diamond$

Here, the empty sum is defined to be zero. An example of a lattice path can be found in Figure 4.6. The following lemma relate runs in $T(\mathcal{A})$ to lattice paths inside $\mathcal{P}_\mathcal{A}(\vec{c})$.

Lemma 4.3.1 *Let $\mathcal{A}$ be a simple bounded one-counter automaton with bound $b \in \mathbb{N}$, there exists a run $\varrho : (q,0) \to_\mathcal{A}^* (q,n)$ with the corresponding path π in $G_\mathcal{A}$ if, and only if, there exists a lattice path ρ starting in $\vec{0}$ and ending in $\vec{c}$ in $\mathcal{P}_\mathcal{A}(\vec{c})$, where $\vec{c} = (f_\pi(e_1), \dots, f_\pi(e_d))$.*

Proof. Let $\varrho : (q,0) \to_\mathcal{A}^* (q,n)$ be a run and let $\pi = e_{i_1} e_{i_2} \dots e_{i_m}$ be its corresponding path in $G_\mathcal{A}$[1]. Define the desired lattice path as $\rho \overset{\text{def}}{=} \vec{u_{i,1}}\vec{u_{i,2}} \dots \vec{u_{i,n}}$. We have $0 \le k \le b$ for all configurations (q,k) visited along ϱ and hence $\sum_{j\in[i]} \vec{u_{i_j}} \in \mathcal{P}_\mathcal{A}(\vec{c})$ for all $i \in [0,m]$. The converse direction follows analogously. $\square$

[1]As we allow for multiple edges between control locations, here a path is a sequence of edges rather than a sequence of vertices.

We remark, but do not prove, that by intersecting $\mathcal{P}_{\mathcal{A}}(\vec{c})$ with the linear flow constraints from the equations (4.1) to (4.4) this lemma can be generalised such that given a path flow of a run in an *arbitrary* bounded one-counter automaton corresponds to the problem of deciding the existence of lattice paths in the corresponding convex polygon.

As stated earlier, in the case of two dimensions it is possible to decide the existence of a lattice path in terms of the discrete volume of $\mathcal{P}_{\mathcal{A}}(\vec{c})$. In the following, let $D(\mathcal{P}) \overset{\text{def}}{=} \#(\mathcal{P} \cap \mathbb{Z}^2)$ denote the *discrete volume* of a polyhedron $\mathcal{P}$.

Lemma 4.3.2 *Let $\mathcal{P}_{\mathcal{A}}(\vec{c})$ be the polyhedron corresponding to a simple bounded one-counter automaton $\mathcal{A}$ with $\vec{c} = (c_1, c_2) \in \mathbb{N}^2$ and let $n = c_1 + c_2$. There exists a lattice path ρ from $\vec{0}$ to $\vec{c}$ in $\mathcal{P}_{\mathcal{A}}(\vec{c})$ if, and only if, $D(\mathcal{P}) \geq n + 1$.*

Proof. ("$\Rightarrow$") Let $\rho = \vec{u_{i,1}} \dots \vec{u_{i,n}}$ be a lattice path in $\mathcal{P}_{\mathcal{A}}(\vec{c})$. We have that $\sum_{k \in [j]} \vec{u_{i,k}} \in \mathcal{P}_{\mathcal{A}}(\vec{c}) \cap \mathbb{Z}^2$ for $j \in [0, n]$ and since each $\vec{u_{i,k}} \neq \vec{0}$ we conclude that $D(\mathcal{P}_{\mathcal{A}}(\vec{c})) \geq n + 1$.

("$\Leftarrow$") If there is a point $(c_1', c_2') \in \mathcal{P}_{\mathcal{A}}(\vec{c})$ such that $(c_1' - 1, c_2') \in \mathcal{P}_{\mathcal{A}}(\vec{c})$ and $(c_1', c_2' - 1) \in \mathcal{P}_{\mathcal{A}}(\vec{c})$ then we have $|a_1| + |a_2| \leq b$. Hence, *any* point in $\mathcal{P}_{\mathcal{A}}(\vec{c})$ has a predecessor in $\mathcal{P}_{\mathcal{A}}(\vec{c})$ and thus every point in $\mathcal{P}_{\mathcal{A}}(\vec{c})$ is reachable via a lattice path starting from the origin.

It remains to consider the case in which each $\vec{c} \in \mathcal{P}_{\mathcal{A}}(\vec{c}) \cap \mathbb{Z}^2$ has at most one predecessor. For $i \in \mathbb{N}$, set $D_i \overset{\text{def}}{=} \{x : (x, i) \in \mathcal{P}_{\mathcal{A}}(\vec{c}) \cap \mathbb{Z}^2\}$. We have $\#(D_i \cap D_{i+1}) \leq 1$ since otherwise some point in $\mathcal{P}_{\mathcal{A}}(\vec{c})$ has two predecessors. Since $D(\mathcal{P}_{\mathcal{A}}(\vec{c})) = \sum_{i \in [0, c_2]} \#D_i \geq c_1 + c_2 + 1$ this implies $\#(D_i \cap D_{i+1}) = 1$ for all $i \in [0, c_2 - 1]$. Thus, every pair of D_i and D_{i+1} contains a point with the same x_1-coordinate, which allows us to construct a unique lattice path from $\vec{0}$ to $\vec{c}$. □

Considering the example in Figure 4.6, we see that the polyhedron $\mathcal{P}_{\mathcal{A}}(\vec{c})$ contains 10 integral points, which by the previous lemma proves the existence of a lattice path from $\vec{0}$ to $(5, 4)$. It remains to show that the discrete volume $D(\mathcal{P}_{\mathcal{A}}(\vec{c}))$ of $\mathcal{P}_{\mathcal{A}}(\vec{c})$ can be computed in polynomial time. Our first observation is that $D(\mathcal{P}_{\mathcal{A}}(\vec{c}))$ can be expressed as a sum of simpler polyhedra. Let us denote by $\triangle(P_1, P_2, P_3)$ and $\triangle(P_1', P_3', P_2')$ the triangles with endpoints P_1, P_2, P_3 respectively P_1', P_3', P_2', by

$\square(P_1', P_3', P_4, P_3)$ the rectangle with endpoints P_1', P_3', P_4, P_3, and by $\overline{P_1P_2}$ and $\overline{P_1'P_2'}$ the lines between P_1 and P_2 respectively P_1' and P_2' in Figure 4.6. Since all endpoints lie at the intersection of linear functions with rational coefficients, all endpoints are rational. The discrete volume of $\mathcal{P}_{\mathcal{A}}(\vec{c})$ can now be expressed as

$$D(\mathcal{P}_{\mathcal{A}}(\vec{c})) = D(\square(P_1', P_3', P_4, P_3)) - D(\triangle(P_1, P_2, P_3)) - D(\triangle(P_1', P_3', P_2')) + D(\overline{P_1P_2}) + D(\overline{P_1'P_2'}). \quad (4.10)$$

We have that $D(\square(P_1', P_3', P_4, P_3)) = (c_1 + 1)(c_2 + 1)$, which clearly is computable in polynomial time. Regarding $D(\overline{P_1P_2})$ and $D(\overline{P_1'P_2'})$ we observe that given a linear function $f(x) = (mx+n)/l$ with $\gcd(m, n, l) = 1$, $f(x) \in \mathbb{Z}$ for $x = kl-n/m$, $k \in \mathbb{Z}$. It follows that the number of lattice points of $f(x)$ in an interval $[x_0, x_1]$ can be obtained by evaluating $\lfloor (x_1 + n/m)/l \rfloor - \lceil (x_0 + n/m)/l \rceil + 1$, which clearly can be computed in polynomial time. Finally, by giving a generalisation of Pick's theorem, Beck shows [7] that the discrete volume of triangles with rational endpoints is computable in polynomial time. Summing up, it follows that $D(\mathcal{P}_{\mathcal{A}}(\vec{c}))$ is computable in polynomial time via Equation (4.10). Hence by Lemma 4.3.2, the existence of a lattice path in $\mathcal{P}_{\mathcal{A}}(\vec{c})$ can be decided in polynomial time.

4.4 Discussion

This chapter established previously unknown complexity results about reachability problems in various classes of counter automata. We showed that reachability in one-counter automata is NP-complete, and, based upon that result, that reachability in parametric one-counter automata is NP-complete as well. For the latter class, we showed an interesting connection with satisfiability in quantifier-free Presburger arithmetic with divisibility. We have also proved that reachability in k-counter automata with no zero tests is undecidable for $k \geq 4$. Finally, we discussed reachability in bounded one-counter automata, which remains an open problem of this thesis.

Our result on reachability in one-counter automata solves a problem left open by Rosier and Yen about boundedness in one-counter automata [95]. It can also be seen to generalise a result by Plandowski and Rytter who show in [89] that deciding membership of compressed unary words in regular expressions with compressed constants is NP-complete. Together with the inter-reducibility result between reachability in two-clock timed automata and bounded one-counter automata in Chapter 3, our result on reachability in one-counter automata also gives some hints that reachability in two-clock timed automata might be in NP. The result on reachability in parametric one-counter automata is closely related to work by Ibarra *et.al.* [65], which shows decidability of reachability for a subset of the class of deterministic parametric one-counter automata with sign tests. The decidability of reachability over the whole class of such automata is stated as an open problem in [65]. Note that although we do not allow negative counter values and sign tests, we allow for nondeterminism. Thus, our result is incomparable with this open problem. From a language-theory perspective, it is interesting to mention that parametric one-counter can generate traces of the form $a^n b^n c^n$, which cannot be generated by pushdown systems [6]. The result that reachability in parametric one-counter automata is decidable has recently been used by Demri and Sangnier in order to show decidability of model checking a fragment of freeze LTL over one-counter automata [41].

With regard to future work, the status of reachability in bounded one-counter automata is a compelling problem that remains to be solved, in particular due to its connection to reachability in two-clock timed automata. It is not clear at the moment whether the lattice-path approach presented in Section 4.3 can be generalised for simple bounded one-counter automata with more than two transitions. If this were the case, employing similar decomposition techniques that have been used in Section 4.1 could then possibly help to solve the general problem. We have seen that determining the complexity of reachability in one-counter automata almost immediately yields decidability of reachability in parametric one-counter automata. It seems conceivable that determining the complexity of reachability in bounded one-counter automata is going to give a similar result for parametric bounded one-counter automata, which in turn might give a result for reachability in parametric two-clock timed automata as discussed at the end of Chapter 3.

A further direction for future work would be to consider synthesis problems for parametric one-counter automata that ensure that no Büchi paths exist in the one-counter automaton obtained under a valuation. There is no obvious way to obtain a solution to this problem via a straightforward adoption of the techniques developed in Section 4.3. Without going into too much detail, in particular QFPAD does not seem to be expressive enough to capture such a problem, since we ask whether there *exists* a valuation to the parameters such that *all* paths are no Büchi paths, which essentially involves one quantifier alternation. Although the first-order theory of $\langle \mathbb{N}, <, +, \cdot, 0, 1 \rangle$ is already undecidable with one $\exists\forall$ quantifier alternation [75], there exist syntactic fragments of the first-order theory of $\langle \mathbb{N}, <, +, \cdot, 0, 1 \rangle$, which are decidable [17]. It should be interesting to investigate whether such a syntactic fragment allows for encoding and solving this synthesis problem.

Chapter 5

Model Checking One-Counter Automata

This chapter is about model checking traces of transition systems generated by one-counter automata and families of transition systems generated by parametric one-counter automata. Model checking a labelled transition system $T = (S, \to, \Lambda, \lambda)$ is to determine whether a formula φ given in some temporal specification logic holds in a state $s \in S$ of T, which we write as $(T, s) \models \varphi$. Generally speaking, a formula holds in a state if the traces that begin in this state are a model of this formula. The precise semantics depend on the type of specification logic under consideration and mainly differ depending on whether we deal with branching-time or linear-time logics. Details are going to be provided in the respective sections of this chapter. In all generality, this chapter is about the computational complexity of the following problems.

OCA Model Checking

INPUT: A one-counter automaton $\mathcal{A}$, a formula φ in some specification logic and $(q, n) \in C(\mathcal{A})$.

QUESTION: Does $(T(\mathcal{A}), (q, n)) \models \varphi$?

For parametric one-counter automata, we are going to consider model checking the family of *all* transition systems generated by *all* instantiations of the parameters.

<u>POCA Model Checking</u>

INPUT: A parametric one-counter automaton $\mathcal{A}$ with parameters Y, a formula φ in some specification logic and $(q, n) \in C(\mathcal{A})$.

QUESTION: Does $(T(\mathcal{A}^\nu), (q, n)) \models \varphi$ for all valuations $\nu : Y \to \mathbb{N}$?

There are a number of computational challenges involved in solving the above problems. First, in general, transition systems generated by one-counter automata have an infinite number of traces leaving from a given state. Second, a parametric one-counter automaton yields an infinite family of one-counter automata since there are infinitely many possible instantiations. It should not be surprising that some model checking problems turn out to be undecidable.

The first part of this chapter deals with the branching-time logics. We are going to consider computation tree logic (CTL) and its syntactic fragment EF and show that CTL model checking of one-counter automata is EXPSPACE-complete, whereas it is PSPACE-complete for EF. For both logics, the model checking problem for parametric one-counter automata is undecidable. The second part then deals with linear-time temporal logic (LTL), for which model checking one-counter automata is PSPACE-complete and coNEXPTIME-complete for parametric one-counter automata. Except for the EF case, the lower bounds are going to be the most interesting. They rely on some number-theoretic encoding of information in counter values, and we are going to construct one-counter automata and formulae in the specification logics under consideration that allow us to access this information in quite sophisticated ways. Where possible, we will give a fine-grained analysis of the computational complexity of the model checking problems and distinguish between their *combined* and their *data complexity.* When establishing the combined complexity of a model checking problem, the counter automaton *and* the formula are allowed to vary. When dealing with the data complexity of a model checking problem, the counter automaton is allowed to vary but the formula is fixed.

Many of the upper bounds provided in this chapter rely on results from the literature on model checking of one-counter automata with updates encoded in unary. We are now going to describe a procedure, which turns a one-counter automaton into a unary one-counter automaton and is going to enable us to make use of those results. The underlying idea is straightforward: we are going to replace every transition updating the counter by a chain of unary updates. All control locations that are newly introduced during this process are going to be labelled with some designated fresh label, which will later allow us to distinguish old from new control locations. More formally, given a one-counter automaton $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda, \xi)$, *the unary one-counter automaton* $\mathcal{A}' = (Q', \Lambda, q_0, F, \Delta', \lambda', \xi')$ *corresponding to* $\mathcal{A}$ is obtained from $\mathcal{A}$ by

- introducing for each transition $(q, q') \in \Delta$ of $\mathcal{A}$ labelled with $\mathsf{add}(z), |z| > 1$, $|z| - 1$ new control locations $(q, q', 1), \ldots, (q, q', |z| - 1) \in Q'$ that are all labelled by λ' with a fresh label α;
- removing the transition (q, q') for any such transition from Δ';
- introducing new transitions $(q, (q, q', 1)), ((q, q', |z|-1), q') \in \Delta'$ and $((q, q', i), (q, q', i+1)) \in \Delta'$ for all $i \in [|z| - 2]$ that are all labelled by ξ' with $\mathsf{add}(+1)$ if $z > 1$ and labelled with $\mathsf{add}(-1)$ if $z < -1$; and
- otherwise leaving $\mathcal{A}$ unchanged.

We have $|\mathcal{A}'| = O(\exp(|\mathcal{A}|))$. Note that $\mathcal{A}'$ can be constructed from $\mathcal{A}$ with a PSPACE transducer.

5.1 Branching-Time Logics

This section considers model checking formulae of the branching-time logic CTL and its syntactic fragment EF on transition systems generated by one-counter automata and families of transition systems generated by parametric one-counter automata. The syntax of CTL is given by the following grammar, where γ ranges over a set of labels Λ:

$$\varphi ::= \gamma \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}(\varphi \mathsf{U} \varphi) \mid \mathsf{E}(\varphi \mathsf{WU} \varphi).$$

The size $|\varphi|$ of a CTL formula φ is the number of symbols required to write it down. We define $\mathsf{true} \overset{\text{def}}{=} \neg(\gamma \wedge \neg\gamma)$ for some $\gamma \in \Lambda$. The additional Boolean connectives are defined as follows: $\varphi_1 \vee \varphi_2 \overset{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \overset{\text{def}}{=} \neg\varphi_1 \vee \varphi_2$ and $\varphi_1 \leftrightarrow \varphi_2 \overset{\text{def}}{=} \varphi_1 \rightarrow \varphi_2 \wedge \varphi_2 \rightarrow \varphi_1$. Moreover, we introduce the following additional modalities: $\mathsf{EF}\varphi \overset{\text{def}}{=} \mathsf{true}\mathsf{U}\varphi$, $\mathsf{AG}\varphi \overset{\text{def}}{=} \neg\mathsf{EF}\neg\varphi$ and $\mathsf{AX}\varphi \overset{\text{def}}{=} \neg\mathsf{EX}\neg\varphi$. The branching-time logic EF is defined as a syntactic fragment of CTL by the following grammar:

$$\varphi ::= \gamma \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{EX}\varphi \mid \mathsf{EF}\varphi.$$

An EF formula is in *negation normal form (NNF)* if all negation symbols only occur in front of a label. Using standard algorithms and the modal abbreviations introduced above, any EF formula can be turned into one in negation normal form in polynomial time.

The semantics of CTL and EF are presented in Table 5.1 and given in terms of a labelled transition system $T = (S, \rightarrow, \Lambda, \lambda)$ and a state $s \in S$.

In [98], Serre shows that modal μ-calculus model checking of unary one-counter automata is decidable and in PSPACE. Since CTL can be embedded into the model μ-calculus [20], it follows that CTL model checking of unary one-counter automata is in PSPACE. We are going to use this result in order to show that CTL model checking of one-counter automata is in $\mathsf{EXPSPACE}$. Given a one-counter automaton $\mathcal{A}$, let $\mathcal{A}'$ be the unary one-counter automaton corresponding to $\mathcal{A}$ as described in the introduction of this chapter. Given an instance $\mathcal{A}, \varphi$ and (q, n) of a CTL model checking problem, the idea is to compute a modified CTL formula $\varphi^\dagger$ such that $(T(\mathcal{A}), (q, n)) \models \varphi$ if, and only if, $(T(\mathcal{A}'), (q, n)) \models \varphi^\dagger$. Informally speaking, the formula $\varphi^\dagger$ needs to ignore states labelled with the fresh proposition α. It is formally defined as follows:

$$\begin{aligned}
\gamma^\dagger &\overset{\text{def}}{=} \gamma & (\mathsf{EX}\varphi)^\dagger &\overset{\text{def}}{=} \mathsf{EX}(\mathsf{E}(\alpha \mathsf{U}(\neg\alpha \wedge \varphi^\dagger))) \\
(\varphi_1 \wedge \varphi_2)^\dagger &\overset{\text{def}}{=} \varphi_1^\dagger \wedge \varphi_2^\dagger & (\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2))^\dagger &\overset{\text{def}}{=} \mathsf{E}((\alpha \vee \varphi_1^\dagger)\mathsf{U}(\neg\alpha \wedge \varphi_2^\dagger)) \\
(\neg\varphi)^\dagger &\overset{\text{def}}{=} \neg\varphi^\dagger & (\mathsf{E}(\varphi_1 \mathsf{WU} \varphi_2))^\dagger &\overset{\text{def}}{=} \mathsf{E}((\alpha \vee \varphi_1^\dagger)\mathsf{WU}(\neg\alpha \wedge \varphi_2^\dagger))
\end{aligned}$$

$$
\begin{aligned}
(T,s) \models \gamma &\iff \gamma \in \lambda(s) \\
(T,s) \models \varphi_1 \wedge \varphi_2 &\iff (T,s) \models \varphi_1 \text{ and } (T,s) \models \varphi_2 \\
(T,s) \models \neg\varphi &\iff (T,s) \not\models \varphi \\
(T,s) \models \mathsf{EX}\varphi &\iff (T,s') \models \varphi \text{ for some } s' \in S \text{ with } s \to s' \\
(T,s) \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) &\iff \text{there are } n \in \mathbb{N}, s_1, \ldots, s_n \in S \text{ such that } s_1 = s, \\
& \qquad s_i \to s_{i+1}, (T,s_i) \models \varphi_1 \text{ for all } i \in [n-1] \\
& \qquad \text{and } (T,s_n) \models \varphi_2 \\
(T,s) \models \mathsf{E}(\varphi_1 \mathsf{WU} \varphi_2) &\iff (T,s) \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \text{ or there are } s_1, s_2, \ldots \in S \\
& \qquad \text{such that } s_1 = s, s_i \to s_{i+1} \text{ and } (T,s_i) \models \varphi_1 \\
& \qquad \text{for all } i \in \mathbb{N}_{>0}
\end{aligned}
$$

Table 5.1: Semantics of CTL.

Note that the size of $\varphi^\dagger$ is linear in the size of φ. The following lemma establishes the correspondence between $\mathcal{A}$ and φ and $\mathcal{A}'$ and $\varphi^\dagger$.

Lemma 5.1.1 *Let $\mathcal{A}$, φ and (q,n) be an instance of an CTL model checking problem, and let $\mathcal{A}'$ and $\varphi^\dagger$ be defined as above. Then $(T(\mathcal{A}),(q,n)) \models \varphi$ if, and only if, $(T(\mathcal{A}'),(q,n)) \models \varphi^\dagger$.*

Proof. We sketch a proof by structural induction on φ and only consider the interesting cases $\varphi = \mathsf{EX}\varphi'$, $\varphi = \mathsf{E}(\varphi_1\mathsf{U}\varphi_2)$ and $\mathsf{E}(\varphi_1\mathsf{WU}\varphi_2)$. For $\varphi = \mathsf{EX}\varphi'$, suppose $(T(\mathcal{A}),(q,n)) \models \varphi$, by the semantic definition there is a (q',n') such that $(q,n) \to_{\mathcal{A}} (q',n')$ and $(T(\mathcal{A}),(q',n')) \models \varphi'$. By the induction hypothesis, $(T(\mathcal{A}'),(q',n')) \models (\varphi')^\dagger$. Moreover, there is a (q,n)-(q',n') path ϱ in $T(\mathcal{A}')$ such that $(T(\mathcal{A}'),\varrho(i)) \models \alpha$ for all $i \in [2,|\varrho|]$. Consequently, $(T(\mathcal{A}'),(q,n)) \models \mathsf{EX}(\mathsf{E}(\alpha\mathsf{U}(\neg\alpha \wedge \varphi^\dagger)))$.

For $\varphi = \mathsf{E}(\varphi_1\mathsf{U}\varphi_2)$, by the semantic definition there exists a (q,n)-(q',n') path ϱ in $T(\mathcal{A})$ such that $(T(\mathcal{A}),(q',n')) \models \varphi_2$ and $(T(\mathcal{A}),\varrho(i)) \models \varphi_1$ for all $i \in [|\varrho|]$. By the induction hypothesis, $(T(\mathcal{A}'),(q',n')) \models \varphi_2^\dagger$ and $(T(\mathcal{A}'),\varrho(i)) \models \varphi_1^\dagger$ for all

$i \in [|\varrho|]$. Moreover, for all $i \in [|\varrho|]$ there are $\varrho(i)$-$\varrho(i+1)$ paths ϱ_i in $T(\mathcal{A}')$ such that $(T(\mathcal{A}'), \varrho_i(j)) \models \alpha$ for all $j \in [2, |\varrho_i|]$. Consequently, $(T(\mathcal{A}'), (q, n)) \models \mathsf{E}((\alpha \vee \varphi_1^\dagger)\mathsf{U}(\neg\alpha \wedge \varphi_2^\dagger))$. The same argument can be adopted to the case of $\varphi = \mathsf{E}(\varphi_1 \mathsf{WU} \varphi_2)$.

The converse direction follows analogously. □

By the results from [98], deciding $(T(\mathcal{A}'), (q, n)) \models \varphi^\dagger$ is in PSPACE in the sum of $|\mathcal{A}'|$, $|\varphi^\dagger|$ and the unary representation of n. Thus, deciding $(T(\mathcal{A}), (q, n)) \models \varphi$ is in $\mathsf{EXPSPACE}$ in the sum of $|\mathcal{A}|$, $|\varphi|$ and the binary representation of n. Moreover, this immediately gives us Π_1^0-membership of CTL model checking on parametric one-counter automata. Given a parametric one-counter automaton $\mathcal{A}$ and a CTL formula φ, in order to decide whether there exists a valuation ν such that $(T(\mathcal{A}^\nu), (q, n)) \not\models \varphi$, we can enumerate all possible valuations ν and check $(T(\mathcal{A}^\nu), (q, n)) \models \varphi$. Enumerating all possible valuations of parameters Y can for example be done by iterating over every $i \in \mathbb{N}$ and considering every valuation ν such that $\sum_{y \in Y} \nu(y) = i$.

Proposition 5.1.1 *Model checking* CTL *on one-counter automata is in* $\mathsf{EXPSPACE}$ *and in* Π_1^0 *for parametric one-counter automata.*

5.1.1 EF Model Checking

We are now going to establish the computational complexity of EF model checking of one-counter and parametric one-counter automata. For one-counter automata with updates encoded in unary, this problem has first been considered by Jančar *et al.* in [68] who established a DP lower bound[1]. A couple of years later, Göller, Mayr and To showed that the problem is actually P^{NP}-complete[2] [51].

The first part of this section considers EF model checking of one-counter automata, where we show that the problem is PSPACE-complete. The more difficult part is showing membership in PSPACE, which is achieved by a thorough periodicity analysis of paths in weighted graphs and builds upon and extends the results from Chapter

[1]A language L is in DP if $L = L_1 \cap L_2$ for languages $L_1 \in \mathsf{NP}$ and $L_2 \in \mathsf{coNP}$.

[2]P^{NP} is the class of problems solvable by a polynomial-time algorithm that has access to an NP oracle.

4. The second part then considers parametric one-counter automata. We are going to show that EF model checking is Π^0_1-complete and provide a lower bound via a reduction from Hilbert's tenth problem.

EF Model Checking of One-Counter Automata

We are now going to show that EF model checking of one-counter automata is PSPACE-complete. For the lower bound, we reduce from validity of quantified Boolean formulae.

Definition 17 A *quantified Boolean formula (QBF)* ψ is a formula of the form

$$\psi = Q_1 x_1 . Q_2 \ldots Q_n x_n . \phi(x_1, \ldots, x_n),$$

where ϕ is a Boolean formula and $Q_i \in \{\exists, \forall\}$ for all $i \in [n]$. $\Diamond$

The *validity of a QBF formula* ψ is defined by induction on n. For $n = 0$, ψ is a propositional formula ϕ, and ψ is valid if ϕ evaluates to *true*. For $n > 0$ and $\psi = Q_1 x_1 . Q_2 x_2 \ldots Q_n x_n . \phi(x_1, \ldots, x_n)$,

- if $Q_1 = \exists$ then ψ is valid if
  - $Q_2 x_2 \ldots Q_n x_n . (\phi[0/x_1])(x_2, \ldots, x_n)$ *or*
  - $Q_2 x_2 \ldots Q_n x_n . (\phi[1/x_1])(x_2, \ldots, x_n)$ are valid; and
- if $Q_1 = \forall$ then ψ is valid if
  - $Q_2 x_2 \ldots Q_n x_n . (\phi[0/x_1])(x_2, \ldots, x_n)$ *and*
  - $Q_2 x_2 \ldots Q_n x_n . (\phi[1/x_1])(x_2, \ldots, x_n)$ are valid.

Checking validity of a quantified Boolean formula is a PSPACE-complete problem [100] and remains PSPACE-hard if we restrict the matrix formula ϕ to be in 3-CNF.

<u>QBF 3-SAT</u>

INPUT: A QBF formula $\psi = Q_1 x_1 \ldots Q_n x_n . \phi(x_1, \ldots, x_n)$ with ϕ in 3-CNF.

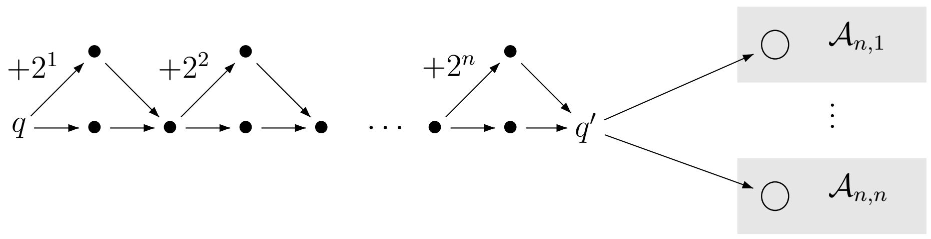

Figure 5.1: One-counter automaton $\mathcal{A}$ constructed in the proof of PSPACE-hardness of EF model checking.

QUESTION: Is ψ valid?

We are now going to show PSPACE-hardness of EF model checking of one-counter automata via a reduction from QBF 3-SAT. Given an instance $\psi = Q_1 x_1 \ldots Q_n x_n.\phi(x_1,\ldots,x_n)$ of QBF 3-SAT, the idea is to encode an assignment of the Boolean variables into the bits of the counter. The EX and AX modalities can then be used in order to simulate the quantifiers of ψ, and the gadgets $\mathcal{A}_{n,i}$ from Section 2.5.2 that allow for testing individual bits of the counter in order to check whether an encoded assignment evaluates ϕ to true.

Proposition 5.1.2 *Model checking* EF *on one-counter automata is* PSPACE*-hard.*

Proof. Let $\psi = Q_1 x_1 \ldots Q_n x_n.\phi(x_1,\ldots,x_n)$ be an instance of QBF 3-SAT. Consider the one-counter automaton $\mathcal{A}$ in Figure 5.1. Starting in q, on any path to q' the automaton can non-deterministically add 2^i to the counter for each $i \in [n]$, where adding 2^i indicates that the Boolean variable x_i is set to 1. The control location q' is then connected to gadgets $\mathcal{A}_{n,1},\ldots,\mathcal{A}_{n,n}$ from the example in Section 2.5.2, which are such that starting in $\bigcirc$, a location labelled with γ_i is reachable in $\mathcal{A}_{n,i}$ if, and only if, the i-th bit of the counter is set to 1, *i.e.*, if 2^i has previously been added to the counter.

We now show how to derive from ψ an EF formula $\psi^\dagger$ such that ψ is valid if, and only if, $(T(\mathcal{A}),(q,0)) \models \psi^\dagger$. For a literal x_i, we define $x_i^\dagger \stackrel{\text{def}}{=} \mathsf{EF}\gamma_i$ and for a literal $\neg x_i$ we set $(\neg x_i)^\dagger \stackrel{\text{def}}{=} \neg x_i^\dagger$. For $\phi = (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge \ldots \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m,3})$, we set $\phi^\dagger \stackrel{\text{def}}{=} (\ell_{1,1}^\dagger \vee \ell_{1,2}^\dagger \vee \ell_{1,3}^\dagger) \wedge \ldots \wedge (\ell_{m,1}^\dagger \vee \ell_{m,2}^\dagger \vee \ell_{m,3}^\dagger)$ and for a quantifier $Q \in \{\exists,\forall\}$, we set $\exists^\dagger \overset{\text{def}}{=} \mathsf{EXEX}$ and $\forall^\dagger \overset{\text{def}}{=} \mathsf{AXAX}$, and finally define $\psi^\dagger \overset{\text{def}}{=} Q_1^\dagger Q_2^\dagger \dots Q_n^\dagger(\phi^\dagger)$. It is easily checked by induction on n that $\psi^\dagger$ has the desired properties. □

We now turn towards showing that EF model checking of one-counter automata is in PSPACE. Decidability of the problem already follows from Proposition 5.1.1, but only gives us an $\mathsf{EXPSPACE}$ upper bound. In order to obtain a PSPACE algorithm, we show a periodicity property of EF formulae. Informally speaking, this is going to allow us to shrink the search space of counter values to consider when trying to prove or disprove formulae of the form $\mathsf{EF}\varphi$, which *a priori* require the inspection of a potentially infinite number of configurations.

To begin with, we state the following lemma. Informally speaking, it says that for every one-counter automaton we can find constants k and δ polynomial in the size of $\mathcal{A}$ and polynomials p, p' such that $p'(k)$ serves as a threshold above which whenever there is a path whose absolute weight is greater than $p(k)$ we can find a path whose weight increases by the period δ. This gives us a periodicity property for reachability relations.

Lemma 5.1.2 *Let $\mathcal{A}$ be a one-counter automaton. There exist constants $k, \delta = O(|\mathcal{A}|^2)$ and fixed polynomials p, p' such that for all $(q, n), (q', n') \in C(\mathcal{A})$,*

(i) if $n - n' > p(k)$ and $n' > p'(k)$ then $(q, n) \to_{\mathcal{A}}^* (q', n')$ if, and only if, $(q, n+\delta) \to_{\mathcal{A}}^* (q', n')$; and

(ii) if $n' - n > p(k)$ and $n > p'(k)$ then $(q, n) \to_{\mathcal{A}}^* (q', n')$ if, and only if, $(q, n) \to_{\mathcal{A}}^* (q', n' + \delta)$.

The proof of this lemma is quite tedious and technical, and we defer it to the end of this section. In the following, let us fix a one-counter automaton $\mathcal{A}$, the constants k, δ and the polynomials p, p' from the Lemma 5.1.2. When proving Lemma 5.1.2, the construction of the polynomial p will ensure that $p(k)$ is greater than the absolute maximum increment in $\mathcal{A}$. This ensures that if there is a run $\varrho : (q, n) \to_{\mathcal{A}}^* (q', n')$ such that $n - n' > p(k)$ then this run can be decomposed as $\varrho = \varrho_1 \cdot \varrho_2$ such that

$\varrho_1 : (q,n) \to^*_{\mathcal{A}} (q'',n'')$, $\varrho_2 : (q'',n'') \to^*_{\mathcal{A}} (q',n')$ and $n - n'' < p(k)$. We will implicitly use this fact in the subsequent lemmas.

We now show the periodicity property for EF formulae: given an EF formula φ, there exists a threshold polynomial in $|\mathcal{A}|$ and $|\varphi|$ such that φ holds with period δ.

Lemma 5.1.3 *Let φ be an EF-formula in negation normal form and $n > p(k)|\varphi| + p'(k)$. Then $(T(\mathcal{A}), (q,n)) \models \varphi$ if, and only if, $(T(\mathcal{A}), (q, n+\delta)) \models \varphi$.*

Proof. We show the statement by structural induction on φ and consider the only interesting cases $\varphi = \mathsf{EX}\varphi'$ and $\varphi = \mathsf{EF}\varphi'$. For $\varphi = \mathsf{EX}\varphi'$, by the semantic definition there exists a configuration (q',n') such that $(q,n) \to_{\mathcal{A}} (q',n')$ and $(T(\mathcal{A}), (q',n')) \models \varphi'$. Since $n' > p(k)|\varphi'| + p'(k)$, the induction hypothesis yields $(T(\mathcal{A}), (q',n')) \models \varphi'$ if, and only if, $(T(\mathcal{A}), (q', n'+\delta)) \models \varphi'$, hence $(T(\mathcal{A}), (q,n)) \models \varphi$ if, and only if, $(T(\mathcal{A}), (q, n+\delta)) \models \varphi$.

If $\varphi = \mathsf{EF}\varphi'$, by the semantic definition there is (q',n') such that there is a run $\varrho : (q,n) \to^*_{\mathcal{A}} (q',n')$ and $(T(\mathcal{A}), (q',n')) \models \varphi'$. Let π be the path corresponding to ϱ in $G_{\mathcal{A}}$. We distinguish two cases: (a) the smallest counter value occurring along ϱ is less than $n - p(k)$ and (b) the smallest counter value occurring along ϱ is at most $n - p(k)$. In the latter case (b), no zero-labelled transition occurs along ϱ. Moreover, $n' > p(k)|\varphi'| + p'(k)$ and hence by the induction hypothesis, $(T(\mathcal{A}), (q',n')) \models \varphi'$ if, and only if, $(T(\mathcal{A}), (q', n'+\delta)) \models \varphi'$. Consequently, we obtain the run $\varrho + \delta : (q, n+\delta)) \to^*_{\mathcal{A}} (q', n'+\delta))$, which gives $(T(\mathcal{A}), (q, n+\delta)) \models \mathsf{EF}\varphi'$. Otherwise in case (a), we can decompose ϱ into $\varrho = \varrho' \cdot \varrho''$ such that $\varrho' : (q,n) \to^*_{\mathcal{A}} (q'',n'')$, $\varrho'' : (q'',n'') \to^* (q',n')$, $n'' > p'(k)$ and $n - n'' > p(k)$. By Lemma 5.1.2(i), we get $(q,n) \to^*_{\mathcal{A}} (q'',n'')$ if, and only if, $(q, n+\delta) \to^*_{\mathcal{A}} (q'',n'')$, hence $(T(\mathcal{A}), (q, n+\delta)) \models \mathsf{EF}\varphi'$. □

We can now provide the EF model checking algorithm. Algorithm 2 is an alternating algorithm that given a one-counter automaton $\mathcal{A}$, a configuration (q,n) and an EF formula φ in negation normal form returns *true* if, and only if, $(T(\mathcal{A}), (q,n)) \models \varphi$. The algorithm proceeds via induction on the structure of φ. The first lines deal with the cases $\varphi = \gamma$, $\varphi = \neg\gamma$, $\varphi = \varphi_1 \wedge \varphi_2$ and $\varphi = \varphi_1 \vee \varphi_2$, which are defined in a

Algorithm 2 EF model checking algorithm deciding $(T(\mathcal{A}),(q,n)) \models \varphi$

Input: One-counter automaton $\mathcal{A}$, EF formula φ in NNF, $(q,n) \in C(\mathcal{A})$

```
case $\varphi = \gamma$: return $\gamma \in \lambda(q)$
case $\varphi = \neg\gamma$: return $\gamma \notin \lambda(q)$
case $\varphi = \varphi_1 \wedge \varphi_2$: return $(T(\mathcal{A}),(q,n)) \models \varphi_1$ && $(T(\mathcal{A}),(q,n)) \models \varphi_2$
case $\varphi = \varphi_1 \vee \varphi_2$: return $(T(\mathcal{A}),(q,n)) \models \varphi_1$ || $(T(\mathcal{A}),(q,n)) \models \varphi_2$
case $\varphi = \mathsf{EX}\varphi'$: existential move:
        $q'$:= choose $\Delta(q)$
        case $\xi(q,q') = \mathsf{add}(z)$ && $n+z \geq 0$: return $(T(\mathcal{A}),(q',n+z)) \models \varphi'$
        case $\xi(q,q') = \mathsf{zero}$ && $n=0$: return $(T(\mathcal{A}),(q',0)) \models \varphi'$
        otherwise: return false
case $\varphi = \mathsf{AX}\varphi'$: universal move:
        $q'$:= choose $\Delta(q)$
        case $\xi(q,q') = \mathsf{add}(z)$ && $n+z \geq 0$: return $(T(\mathcal{A}),(q',n+z)) \models \varphi'$
        case $\xi(q,q') = \mathsf{zero}$ && $n=0$: return $(T(\mathcal{A}),(q',0)) \models \varphi'$
        otherwise: return true
case $\varphi = \mathsf{EF}\varphi'$: existential move:
        $q'$ := choose $\Delta(q)$
        $n'$ := choose $\{m \in \mathbb{N} : m \leq \max\{n+2p(k), p(k)|\varphi'|\} + p'(k) + \delta + 1\}$
        existential move:
                if $(q,n) \rightarrow^*_{\mathcal{A}} (q',n')$ then return $(T(\mathcal{A}),(q',n')) \models \varphi'$
                else return false
case $\varphi = \mathsf{AG}\varphi'$: universal move:
        $q'$ := choose $\Delta(q)$
        $n'$ := choose $\{m \in \mathbb{N} : m \leq \max\{n+2p(k), p(k)|\varphi'|\} + p'(k) + \delta + 1\}$
        existential move:
                if $(q,n) \rightarrow^*_{\mathcal{A}} (q',n')$ then return $(T(\mathcal{A}),(q',n')) \models \varphi'$
                else return true
```

straightforward way by just expanding the semantic definition of EF formulae. The cases $\varphi = \mathsf{EX}\varphi'$ or $\varphi = \mathsf{AX}\varphi'$ are also straightforward. In the former case, a successor configuration (q', n') of (q, n) is non-deterministically chosen, and Algorithm 2 then recursively determines $(T(\mathcal{A}), (q', n')) \models \varphi'$. In the latter case, the algorithm checks $(T(\mathcal{A}), (q', n')) \models \varphi'$ for all possible successor configurations (q', n') of (q, n). There are two cases remaining: $\varphi = \mathsf{EF}\varphi'$ and $\varphi = \mathsf{AG}\varphi'$. For $\varphi = \mathsf{EF}\varphi'$, the algorithm non-deterministically chooses a configuration (q', n'), where

$$n' \leq \max\{n + 2p(k), p(k)|\varphi'|\} + p'(k) + \delta + 1$$

It then checks whether $(q, n) \to^*_{\mathcal{A}} (q', n')$, recursively checks $(T(\mathcal{A}), (q', n')) \models \varphi'$ and returns the result. The algorithm handles the case $\varphi = \mathsf{AG}\varphi'$ analogously, but checks $(T(\mathcal{A}), (q', n')) \models \varphi'$ for all (q', n') with n' as above such that $(q, n) \to^*_{\mathcal{A}} (q', n')$. It is clear that Algorithm 2 runs in alternating polynomial time, in particular since the quantification in the EF and AG case is over a set of elements of size polynomial in the size of $\mathcal{A}$, n and φ. Hence by Theorem 2.4.1, the algorithm runs in PSPACE. The correctness of the algorithm is immediate for all cases except for the EF and AG modalities.

Proposition 5.1.3 *Given a one-counter automaton $\mathcal{A}$, $(q, n) \in C(\mathcal{A})$ and an EF formula φ, Algorithm 2 decides $(T(\mathcal{A}), (q, n)) \models \varphi$ in PSPACE.*

Proof. As discussed above, the algorithm runs in alternating polynomial time and hence in PSPACE.

Correctness of the algorithm is shown by induction on the structure of φ. As already discussed, we only consider the cases $\varphi = \mathsf{EF}\varphi'$ and $\varphi = \mathsf{AG}\varphi'$. Suppose $(T(\mathcal{A}), (q, n)) \models \mathsf{EF}\varphi'$, by the semantic definition there exists (q', n') such that $(q, n) \to^*_{\mathcal{A}} (q', n')$ and $(T(\mathcal{A}), (q', n')) \models \varphi'$. In order to prove Algorithm 2 correct, we need to show that there is $n'' \leq \max\{n + 2p(k), p(k)|\varphi'|\} + p'(k) + \delta + 1$ such that $(q, n) \to^*_{\mathcal{A}} (q', n'')$ and $(T(\mathcal{A}), (q', n'')) \models \varphi'$. If $n' > \max\{n+2p(k), p(k)|\varphi'|\}+p'(k)+\delta + 1$, it follows from Lemma 5.1.3 that $(T(\mathcal{A}), (q, m)) \models \varphi'$ for all

$$m \in \{n' + i\delta : i \in \mathbb{Z}\} \cap \{m' \in \mathbb{N} : m' > p(k)|\varphi| + p'(k)\}.$$

Moreover, we have $(q,n) \to_{\mathcal{A}}^{*} (q',m)$ for all

$$m \in \{n' + i\delta : i \in \mathbb{Z}\} \cap \{m' \in \mathbb{N} : m' > n + 2p(k) + p'(k)\}.$$

The latter fact follows directly or indirectly from Lemma 5.1.2(ii): if $n \geq p'(k)$ then we can directly apply Lemma 5.1.2(ii). Otherwise, there is a run $\varrho : (q,n) \to_{\mathcal{A}}^{*} (q'',m')$ such that $m' > p'(k)$, $n' - m' > p(k)$ and $(q'',m') \to_{\mathcal{A}}^{*} (q',n')$. It follows that the required n'' exists.

The case $\varphi = \mathsf{AG}\varphi'$ follows analogously to the EF case. $\square$

It remains to prove Lemma 5.1.2. The proof is split into several smaller lemmas and begins with showing periodicity properties of weights of paths in weighted graphs. Let us fix a weighted graph $G = (V, E, \mu)$. We define a period δ, which is going to be the least common multiple of the greatest common divisors of all weights of all simple cycles in all strongly connected components of G. Formally, given a strongly connected component S of G, we define

$$M(S) \stackrel{\text{def}}{=} \{z \in \mathbb{Z} \setminus \{0\} : \text{there is a simple } v\text{-cycle } \ell \text{ s.t. } v \in S \text{ and } \mathit{weight}(\ell) = z\} \text{ and}$$

$$\gcd(S) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } M(S) = \emptyset \\ \gcd(M(S)) & \text{otherwise} \end{cases}$$

The period δ is now defined as

$$\delta \stackrel{\text{def}}{=} \operatorname{lcm}(\{\gcd(S) : S \text{ is a strongly connected component in } G\}).$$

Let $r \in \mathbb{N}$ be the absolute maximum value of all weights occurring as labels in G and let $k \stackrel{\text{def}}{=} r|V|$, *i.e.*, the maximum absolute value that the weight of a simple cycle in G can be. Since $1 \leq \gcd S \leq k$ for any strongly connected component S, we have $1 \leq \delta \leq |V|k$. Thus, the binary representation of δ is $O(|G|^3)$.

In order to show a periodicity property of weights of arbitrary paths in weighted graphs, we are first going to establish a periodicity property for cycles. Recall that $S(v)$ denotes the strongly connected component of v, our first observation is that $\gcd(S(v))$ divides the weight of any v-cycle.

Lemma 5.1.4 *Let S be a strongly connected component, $v \in S$ and ℓ be a v-cycle such that $g = \gcd(S)$ and $z = weight(\ell)$, then $g|z$.*

Proof. We show the statement by induction on $|\ell|$. For the induction step, let ℓ' be obtained from ℓ by removing some simple w-cycle ℓ_w from ℓ for some $w \in S$. We have $z = weight(\ell') + weight(\ell_w)$. By the induction hypothesis, $g|weight(\ell')$ and the definition of $\gcd(S)$ yields $\gcd(S)|weight(\ell_w)$, hence $g|z$. $\square$

The converse is however not true in general. For example, if there were exactly two simple v-cycles in S with weights 5 and 7 say, no v-cycle can have weight 23 even though $\gcd(5, 7) = 1$ divides 23. Determining which weights can be achieved by some cycles relates to the Frobenius problem for which the following result is known. For a proof, see *e.g.* [99].

Proposition 5.1.4 *Let $a_1 < \ldots < a_m \in \mathbb{N}_{>0}$, $g = \gcd\{a_1, \ldots, a_m\}$ and $p(\vec{x}) = a_1x_1 + \ldots + a_mx_m$ be a linear polynomial with constant term zero. The set*

$$M = \{z \in \mathbb{Z} : \textit{there exists } \vec{n} \in \mathbb{N}^m \textit{ such that } z = p(\vec{n})\}$$

can be written as $M = U \cup a + g\mathbb{N}$, where $U \subseteq [0, a_m^2]$ and $a = \min\{n \in \mathbb{N} : n \geq a_m^2$ and $g|n\}$.

In the above example, this proposition implies that for any $n \geq 49$ there is a v-cycle ℓ such that $weight(\ell) = n$. Since simple cycles in strongly connected components can have mixed positive and negative weights, we need a slightly more general version of this proposition for our purposes.

Lemma 5.1.5 *Let $a_1 < \ldots < a_m \in \mathbb{Z}$, $n = \max_{i\in[m]}\{|a_i|\}$, $g = \gcd\{a_1, \ldots, a_m\}$, and let $a, p(\vec{x})$ and M be defined as in Proposition 5.1.4. Then*

(i) if $a_1 > 0$ then $M = U \cup a + g\mathbb{N}$ for some $U \subseteq [0, n^2]$;

(ii) if $a_m < 0$ then $M = U \cup -a - g\mathbb{N}$ for some $U \subseteq [-n^2, 0]$; and

(iii) if $a_1 < 0$ and $a_m > 0$ then $M = g\mathbb{Z}$.

Proof. (i) This case follows immediately from Proposition 5.1.4. (ii) Define $a'_i \overset{\text{def}}{=} -a_i$ for all $i \in [k]$. We have that $p'(\vec{x}) = a'_1 x_1 + \ldots + a'_m x_m$ is an instance of case (i) with integral solutions M' and $M = \{-z' : z' \in M'\} = \{-u' : u' \in U'\} \cup -a - g\mathbb{N}$. (iii) By Euclid's theorem, there exists $\vec{y} = (y_1, \ldots, y_m) \in \mathbb{Z}^m$ such that $g = p(\vec{y})$. For any $z \in M$, by setting $\vec{y'} \overset{\text{def}}{=} (y_1 z/g, \ldots, y_m z/g)$, we have $z = p(\vec{y'})$. We now show how to obtain $\vec{n} \in \mathbb{N}^m$ from $\vec{y'}$ such that $z = p(\vec{n})$. To this end, we iterate the following process: for any a_i such that $a_i < 0$ and $y'_i < 0$, let $x_i \in \mathbb{N}$ be chosen such that $y'_i + x_i a_m \geq 0$ and replace y'_i with $y'_i + x_i a_m$ and y_m with $y'_m + x_i |a_i|$. Once $y'_i \geq 0$ for all i such that $a_i < 0$, we turn towards those a_i for which $a_i > 0$ and $y'_i < 0$. For any such a_i, let x_i be chosen such that $y'_i + x_i |a_1| \geq 0$. We replace y'_i with $y_i + x_i |a_1|$ and y_1 with $y_1 + x_i a_i$. After this process has finished, we obtain a vector $\vec{n} \in \mathbb{N}^m$ such that $p(\vec{n}) = p(\vec{y'})$. □

The previous lemmas indicate that once we cross a certain threshold, the converse direction of Lemma 5.1.4 begins to hold. We are now going to make this intuition formal. Recall that $G = (V, E, \mu)$ is a fixed weighted graph and let k be defined as above, *i.e.*, the maximum absolute weight that can be achieved on a simple cycle in G, and let δ be the period as defined above. We define three fixed polynomials p_1, p_2, p_3 that yield bounds for the periodicity properties that we aim to establish. The intention behind the polynomials is as follows: whenever there is a v-cycle ℓ such that $\mathit{weight}(\ell) \leq -p_2(k)$ we can construct a v-cycle ℓ' such that $\mathit{weight}(\ell') = \mathit{weight}(\ell) - \delta$, which yields the desired periodicity property. The analogue relationship holds for cycles with positive weight. Building upon this result, we then show that for an arbitrary v-w path π with $\mathit{weight}(\pi) \leq -p_3(k)$, we can construct a v-w path π' such that $\mathit{weight}(\pi') = \mathit{weight}(\pi) - \delta$. In both cases, the drop of ℓ' and π' does not decrease by more than $p_1(k)$. The three polynomials are defined as follows:

$$p_1(k) \overset{\text{def}}{=} 2k^2 + k \qquad p_2(k) \overset{\text{def}}{=} 3k^2 + k \qquad p_3(k) \overset{\text{def}}{=} (k+1)(p_2(k) + \delta)$$

We are now going to show the periodicity property for cycles and paths with negative weights and then use symmetry to lift our results to cycles and paths with positive weights.

Lemma 5.1.6 *Let $v \in V$ and $g = \gcd(S(v))$. For any $z \in \mathbb{Z}$ such that $z \leq -p_2(k)$,*

(i) if there exists a v-cycle ℓ with weight$(\ell) = z$ then there exists a v-cycle ℓ' with weight$(\ell') = z - \delta$ and drop$(\ell') \geq z - \delta - p_1(k)$; and

(ii) if there exists a v-cylce ℓ with weight$(\ell) = z - \delta$ then there exists a v-cycle ℓ' with weight$(\ell') = z$ and drop$(\ell') \geq z - p_1(k)$.

Proof. We show (i), statement (ii) follows analogously. Let $\ell_1, \ldots, \ell_j$ be all simple v_i-cycles in the strongly connected component of v. We have $\ell_i \in [-k, k]$ for all ℓ_i, and hence $j \in [2k]$. Let π_i denote a simple v-cycle that traverses v_i, *i.e.*, $\pi_i = \pi_{i,1} \cdot \pi_{i,2}$ for a simple v-v_i path $\pi_{i,1}$ and a v_i-v path $\pi_{i,2}$. Since $j \in [2k]$, we have $\sum_{i \in [j]} \mathit{weight}(\pi_i) \geq -2k^2$ and hence $m \overset{\text{def}}{=} z - \sum_{i \in [j]} \mathit{weight}(\pi_i) \leq -k^2 - k$. By Lemma 5.1.4, we have $\gcd(S(v)) | \delta$ and hence Lemma 5.1.5 guarantees the existence of $a_1 \ldots, a_j \in \mathbb{N}$ such that

$$m - \delta = a'_1 \mathit{weight}(\ell_1) + \ldots + a'_j \mathit{weight}(\ell_j).$$

With no loss of generality, we assume that the weights of the ℓ_i are ordered, *i.e.*, $\mathit{weight}(\ell_1), \ldots, \mathit{weight}(\ell_g) < 0 < \mathit{weight}(\ell_{g+1}), \ldots, \mathit{weight}(\ell_j)$ for some $g \in [j]$. Set $\ell' \overset{\text{def}}{=} \ell_u \cdot \ell_d$, where

$$\begin{aligned} \ell_u &\overset{\text{def}}{=} \pi_{g+1,1} \cdot {\ell_{g+1}}^{a'_{g+1}} \cdot \pi_{g+1,2} \cdot \ldots \cdot \pi_{j,1} \cdot {\ell_j}^{a'_j} \cdot \pi_{j,2} \\ \ell_d &\overset{\text{def}}{=} \pi_{1,1} \cdot {\ell_1}^{a'_1} \cdot \pi_{1,2} \cdot \ldots \cdot \pi_{g,1} \cdot {\ell_g}^{a_g} \cdot \pi_{g,2}. \end{aligned}$$

Thus, ℓ_u traverses all positive cycles and ℓ_d all negative cycles. Clearly, $\mathit{weight}(\ell') = z - \delta$. Regarding the drop of ℓ', since all loops in ℓ_u are positive, we have $\mathit{drop}(\ell_u) \geq -2k^2 - k$, as this is the minimum total weight that the π_i can take. The same argument yields $\mathit{drop}(\ell_d^\dagger) \geq -2k^2 - k$. In order to estimate $\mathit{drop}(\ell')$, using Lemma 4.1.2, we conclude that

$$\begin{aligned} \mathit{drop}(\ell') &= \mathit{drop}(\ell_u \cdot \ell_d) \\ &= \min\{\mathit{drop}(\ell_u), \mathit{weight}(\ell_u) + \mathit{drop}(\ell_d)\} \\ &= \min\{\mathit{drop}(\ell_u), \mathit{weight}(\ell_u) + \mathit{weight}(\ell_d) + \mathit{drop}(\ell_d^\dagger)\} \\ &\geq \mathit{weight}(\ell') - 2k^2 - k. \end{aligned}$$

Consequently, $\mathit{drop}(\pi') \geq z - \delta - p_1(k)$. □

As stated earlier, building upon this lemma, we can now show a similar statement for *arbitrary paths* with negative weight that exceeds $-p_3(k)$.

Lemma 5.1.7 *Let $v, w \in V$. For all $z \in \mathbb{Z}$ such that $z \leq -p_3(k)$,*

(i) if there exists a v-w path π with $\mathit{weight}(\pi) = z$ then there exists a v-w path π' with $\mathit{weight}(\pi') = z - \delta$ and $\mathit{drop}(\pi') \geq \min\{\mathit{drop}(\pi) - \delta, z - \delta - p_1(k)\}$; and

(ii) if there exists a v-w path π with $\mathit{weight}(\pi) = z - \delta$ then there exists a v-w π' with $\mathit{weight}(\pi') = z$ and $\mathit{drop}(\pi') \geq \min\{\mathit{drop}(\pi) + \delta, z - p_1(k)\}$.

Proof. We show statement (ii). Statement (i) can be shown in a similar way. Suppose there exists a v-w path π with $\mathit{weight}(\pi) = z - \delta$. We divide z into $k+1$ evenly sized segments, so each such segment has size at least $p_2(k) - \delta$. Since $|V| \leq k$, it thus follows that we can write π as $\pi = \pi_1 \cdot \ell \cdot \pi_2$ such that π_1 is a v-v' path, $\mathit{drop}(\pi_1) \geq z - \delta$, ℓ is a v'-cycle for some $v' \in V$, $\mathit{weight}(\ell) \leq -p_2(k) - \delta$, $\mathit{weight}(\pi_1 \cdot \ell) \geq z$ and π_2 is a v'-w path. Lemma 5.1.6 yields a v'-cycle ℓ' with $\mathit{weight}(\ell') = \mathit{weight}(\ell) + \delta$ and $\mathit{drop}(\ell') \geq \mathit{weight}(\ell) + \delta - p_1(k)$. We define π' as $\pi' \overset{\text{def}}{=} \pi_1 \cdot \ell' \cdot \pi_2$. Clearly, $\mathit{weight}(\pi') = \mathit{weight}(\pi) + \delta = z$. Regarding the drop of π', using the identities from Lemma 4.1.2, we have

$$\begin{aligned}
\mathit{drop}(\pi') &= \mathit{drop}(\pi_1 \cdot \ell' \cdot \pi_2) \\
&= \min\{\mathit{drop}(\pi_1), \mathit{weight}(\pi_1) + \mathit{drop}(\ell' \cdot \pi_2)\} \\
&= \min\{\mathit{drop}(\pi_1), \mathit{weight}(\pi_1) + \mathit{drop}(\ell'), \mathit{weight}(\pi_1 \cdot \ell') + \mathit{drop}(\pi_2))\} \\
&\geq \min\{z, z - p_1(k), \mathit{drop}(\pi) + \delta\} \\
&= \min\{z - p_1(k), \mathit{drop}(\pi) + \delta\}
\end{aligned}$$

Using the symmetry between G and $G^\dagger$, we now obtain a corresponding result for paths with positive weight exceeding $p_3(k)$.

Lemma 5.1.8 *Let $v, w \in V$. For all $z \in \mathbb{Z}$ such that $z \geq p_3(k)$,* *(i) if there exists a v-w path π with weight$(\pi) = z$ then there exists a v-w path π with weight$(\pi') = z + \delta$ and drop$(\pi') \geq \min\{-p_1(k), drop(\pi)\}$; and*

(ii) if there exists v-w path π with weight$(\pi) = z + \delta$ then there exists a v-w path π' with weight$(\pi') = z$ and drop$(\pi') \geq \min\{-p_1(k), drop(\pi)\}$.

Proof. We only show statement (i), the other statement follows along similar lines. Suppose there exists a v-w path π with $weight(\pi) = z$. We have $weight(\pi^\dagger) = -z \leq -p_3(k)$. By Lemma 5.1.7, there exists a path $(\pi')^\dagger$ such that $weight((\pi')^\dagger) = -z - \delta$ and $drop((\pi')^\dagger) \geq \min\{drop(\pi^\dagger) - \delta, -z - \delta - p_3(k)\}$. Hence, $weight(\pi') = z + \delta$ as required. Regarding the drop of π', again by using the identities from Lemma 4.1.2, we conclude

$$\begin{aligned} drop(\pi') &= weight(\pi') + drop((\pi')^\dagger) \\ &\geq z + \delta + \min\{drop(\pi^\dagger) - \delta, -z - \delta - p_1(k)\} \\ &= \min\{drop(\pi), -p_1(k)\}. \end{aligned}$$

We can now use the results obtained for paths in weighted graphs in order to prove Lemma 5.1.2. Let us fix a one-counter automaton $\mathcal{A}$ and let $G_\mathcal{A}$ be the weighted graph corresponding to $\mathcal{A}$. As a slight adjustment, we assume that $G_\mathcal{A}$ is such that every zero-labelled transition in $\mathcal{A}$ is replaced with a transition labelled with "+0" in $G_\mathcal{A}$. The constants k and δ are defined as above.

Lemma 5.1.9 *Let $n, n' \in \mathbb{N}$,*

(i) if $n - n' > p_3(k) + k$ and $n' > p_1(k)$ then $(q, n) \to_\mathcal{A}^* (q', n')$ if, and only if, $(q, n + \delta) \to_\mathcal{A}^* (q', n')$; and

(ii) if $n' - n > p_3(k) + k$ and $n > p_1(k)$ then $(q, n) \to_\mathcal{A}^* (q', n')$ if, and only if, $(q, n) \to_\mathcal{A}^* (q', n' + \delta)$.

Proof. (i) Suppose there is a run $\varrho : (q, n) \to_\mathcal{A}^* (q', n')$ and let π be the corresponding q-q' path in $G_\mathcal{A}$. We can write π as $\pi = \pi' \cdot \pi_0$ such that $\pi' : q \to_{G_\mathcal{A}}^* q''$, $weight(\pi') <$

$-p_3(k)$ and $drop(\pi') \geq -p_3(k) - k$. By Lemma 5.1.7(i), there exists a path $\pi'' : q \to^*_{G_\mathcal{A}} q''$ such that $weight(\pi'') = weight(\pi') - \delta$ and $drop(\pi'') \geq weight(\pi') - \delta - p_1(k)$. Consequently, there exists a run $(q, n + \delta) \to^*_\mathcal{A} (q', n')$ whose corresponding path is $\pi'' \cdot \pi_0$. The converse direction follows in similar way by applying Lemma 5.1.7(ii).

(ii) Suppose there is a run $(q, n) \to^*_\mathcal{A} (q', n')$ and let π be the corresponding q-q' path in $G_\mathcal{A}$. We can write π as $\pi = \pi_0 \cdot \pi'$ such that $\pi' : q'' \to^*_{G_\mathcal{A}} q'$, $weight(\pi') > p_3(k)$ and $drop(\pi') \geq -p_1(k)$. By Lemma 5.1.8(i), there exists a q''-q' path π'' such that $weight(\pi'') = weight(\pi') + \delta$ and $drop(\pi'') \geq -p_1(k)$. Consequently, there is a run $(q, n) \to_\mathcal{A} (q', n' + \delta)$ whose corresponding path is $\pi_0 \cdot \pi''$. The converse direction follows in a similar way by applying Lemma 5.1.8(ii). □

Lemma 5.1.2 now follows as a direct consequence of Lemma 5.1.9 by defining k and δ as above and setting $p(k) \overset{\text{def}}{=} p_3(k) + k$ and $p'(k) \overset{\text{def}}{=} p_1(k)$. Taking together Propositions 5.1.2 and 5.1.3, we obtain the main result of this section.

Theorem 5.1.1 *Model checking* EF*-formulae on parametric one-counter automata is* PSPACE*-complete.*

EF Model Checking of Parametric-One Counter Automata

We now consider model checking EF formulae on parametric one-counter automata and show that this problem is Π^0_1-complete. Hardness for Π^0_1 is shown via a reduction from Hilbert's tenth problem which we sketch in the following. Recall that Hilbert's tenth problem is to decide for a given polynomial $p : \mathbb{R}^n \to \mathbb{R}$ whether there are $a_1, \ldots, a_n \in \mathbb{Z}$ such that $p(a_1, \ldots, a_n) = 0$. As discussed in Chapter 2, it is actually sufficient to restrict the a_i to be from $\mathbb{N}$. For our reduction, the crucial step is to show how we can express a multiplication relation over the parameters of a parametric one-counter automaton. To this end, we use a trick that became popular by the work of Robinson [94]: multiplication can be defined in terms of the least common multiple.

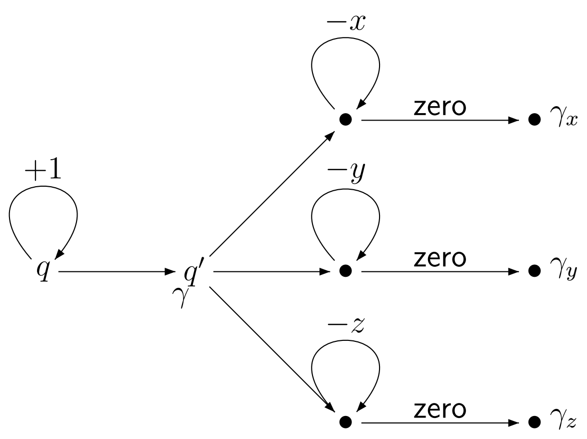

Figure 5.2: The parametric one-counter automaton $\mathcal{A}_{lcm}$ used for testing whether $\nu(z) = \mathrm{lcm}(\nu(x), \nu(y))$ for a given valuation ν.

In fact, given $x, y \in \mathbb{N}$, we have

$$\begin{aligned} & \mathrm{lcm}(x+y, x+y+1) - \mathrm{lcm}(x, x+1) - \mathrm{lcm}(y, y+1) \\ = {} & (x^2 + x + 2xy + y^2 + y) - (x^2 + x) - (y^2 + y) \qquad (*) \\ = {} & 2xy. \end{aligned}$$

The multiplication relation in a parametric one-counter automaton can now be expressed as follows.

Lemma 5.1.10 *There exists a fixed parametric one-counter automaton $\mathcal{A}_{mul}$ with parameters x, y, z and a control location q and a fixed* EF *formula φ such that for any valuation ν, $(T(\mathcal{A}^\nu), (q, 0)) \models \varphi$ if, and only if, $\nu(z) = \nu(x)\nu(y)$.*

Proof. In order to construct $\mathcal{A}_{mul}$, we make use of the identities in $(*)$ and first construct a parametric one-counter automaton $\mathcal{A}_{lcm}$ with parameters x, y, z, a control location q and an EF formula φ_{lcm} such that for any valuation ν, $(T(\mathcal{A}^\nu_{lcm}), (q, 0)) \models \varphi_{lcm}$ if, and only if, $\nu(z) = \mathrm{lcm}(\nu(x), \nu(y))$. For any $r, s, t \in \mathbb{N}$, we have that $t = \mathrm{lcm}(r, s)$ if, and only if, for all $a \in \mathbb{N}$,

$$t | a \Leftrightarrow \mathrm{lcm}(r, s) | a \Leftrightarrow (rs/\gcd(r, s)) | a \Leftrightarrow r | a \text{ and } s | a.$$

Now consider the parametric one-counter automaton $\mathcal{A}_{lcm}$ in Figure 5.2 and the EF formula

$$\varphi = \mathsf{AG}(\gamma \rightarrow ((\mathsf{EF}\gamma_x \wedge \mathsf{EF}\gamma_y) \leftrightarrow (\mathsf{EF}\gamma_z))).$$

Suppose that ν is a valuation such that $(T(\mathcal{A}^{\nu}_{lcm}), (q, 0)) \models \varphi$, then φ enforces for all $n \in \mathbb{N}$ that $\nu(x)|n$ and $\nu(y)|n$ if, and only if, $\nu(z)|n$. Hence by the above reasoning, $\nu(z) = \text{lcm}(\nu(x), \nu(y))$.

In order to construct the required parametric one-counter automaton $\mathcal{A}_{mul}$, it is thus sufficient to introduce additional slack parameters x_1, x_2, x_3, ensure via gadgets $\mathcal{A}_{lcm}$ that $x_1 = \text{lcm}(x+y, x+y+1)$, $x_2 = \text{lcm}(x, x+1)$, $x_3 = \text{lcm}(y, y+1)$ and assert that $2z = x_1 - x_2 - x_3$. $\square$

Being able to express multiplication relations between parameters of a parametric one-counter automaton immediately enables us reduce any instance of Hilbert's tenth problem to an instance of an EF model checking problem, which thus implies Π^0_1-hardness of the latter problem. Due to the existence of a universal polynomial, it follows that there is a parametric one-counter automaton with a fixed number of control locations and parameters and a fixed EF formula for which the model checking problem is undecidable. Since EF is a notational fragment of CTL, membership in Π^0_1 follows from Proposition 5.1.1.

Theorem 5.1.2 *Model checking* EF*-formulae on parametric one-counter automata is* Π^0_1*-complete already for a parametric one-counter automaton with a fixed number of control locations and a fixed* EF*-formula.*

5.1.2 Computation Tree Logic (CTL) Model Checking

This section deals with model checking formulae of CTL on one-counter and parametric one-counter automata. In the introduction, we have already stated in Proposition 5.1.1 that this problem is in $\mathsf{EXPSPACE}$ for one-counter and in Π^0_1 for parametric one-counter automata. Subsequently, we are going to show that those bounds are tight. Although Π^0_1-hardness of CTL model checking on parametric one-counter automata already follows from Π^0_1-hardness of EF model checking, we strengthen this result by showing that CTL model checking is already Π^0_1-hard for parametric one-counter automata with only one parameter.

CTL Model Checking of One-Counter Automata

In this section, we are going to show that CTL model checking of transition systems generated by one-counter automata is EXPSPACE-complete. As discussed above, membership in EXPSPACE already follows from Proposition 5.1.1. We therefore concentrate on the lower bound and show that the problem is EXPSPACE-hard for a fixed CTL formula.

Proving PSPACE-hardness of modal μ-calculus model checking of one-counter automata with updates encoded in unary can be achieved via a straightforward reduction from the halting problem of an alternating polynomial-time Turing machine acting on a unary alphabet, which is known to be PSPACE-complete [61, 69]. However, PSPACE-hardness of CTL model checking of those automata requires some more effort and was shown by Göller and Lohrey in [50]. Instead of directly reducing from the halting problem of a PSPACE Turing machine, in [50] the authors make use of logspace-serialisability of PSPACE in order to obtain PSPACE-hardness. Inspired by the ideas from [50], we are subsequently going to show EXPSPACE-hardness of CTL model checking of one-counter automata by making use of the fact that EXPSPACE is exponentially logspace-serialisable as defined in Section 2.4.1. Although we are going to provide a large amount of technical details, for the sake of understandability and readability we only sketch our reduction, *i.e.*, show the existence of a reduction proving EXPSPACE-hardness.

Before we begin with the reduction, we need to introduce some additional notation and results that are concerned with an alternative way of representing natural numbers. Given naturals $m, n \in \mathbb{N}$, the *Chinese remainder representation* $\mathrm{CRR}_m(n)$ of n is a word over the alphabet $\{0, 1\}$ and defined as

$$\mathrm{CRR}_m(n) \stackrel{\text{def}}{=} (b_{i,0} \cdots b_{i,p_i-1})_{i \in [m]},$$

where p_i is the i-th prime number and $b_{i,j} \overset{\text{def}}{=} 1$ if $n \equiv j \mod p_i$ and $b_{i,j} \overset{\text{def}}{=} 0$ otherwise. The following problem is concerned with obtaining a bit of a number given in Chinese remainder representation.

<u>BIT-FROM-CRR</u>

INPUT: $\text{CRR}_m(n)$ of some $n, m \in \mathbb{N}$, $i \in [0, m-1]$ and $b \in \{0, 1\}$.
QUESTION: Is $\text{bit}_i(n \mod 2^m) = b$?

The following proposition states that BIT-FROM-CRR is computable in logarithmic space. It is a consequence of the result that division is computable in logspace-uniform NC^1[3], which was shown in [26], Theorem 3.3.

Proposition 5.1.5 ([26]) BIT-FROM-CRR *is computable in* L.

We are now going to prove $\mathsf{EXPSPACE}$-hardness of CTL model checking of one-counter automata. Given a language $L \in \mathsf{EXPSPACE}$, by Theorem 2.4.2 there exists a regular language R such that L is exponentially L-serialisable via R. Hence, there exists an L-Turing machine $\mathcal{M}$ and a polynomial p such that for any $w \in \{0,1\}^*$ and $m = \exp^2(p(|w|))$, $w \in L$ if, and only if, $\big(\chi_{\mathcal{M}}(w \cdot \text{bin}_{(\lg m)}(d))\big)_{d \in [0, m-1]} \in R$. Suppose we were asked to write a program that decides $w \in L$ via the serialisation of L. A possible solution to this task is given by Algorithm 3. It requires w and the serialisation of L, *i.e.*, an L Turing machine $\mathcal{M}$, a regular language R given as a deterministic finite state automaton $\mathcal{A}_R$ and a polynomial p as input. For m as above, the algorithm successively iterates through all $d \in [0, m-1]$ and simulates in each iteration $\mathcal{A}_R$ on input $\chi_{\mathcal{M}}(w \cdot \text{bin}_{(\lg m)}(d))$. Once $d = m$, Algorithm 3 returns *true* if the simulation of $\mathcal{A}_R$ ended in an accepting state and *false* otherwise. Our goal in this section is to simulate an execution of Algorithm 3 via an instance of a CTL model checking problem of some polynomial-time computable one-counter automaton $\mathcal{A}_L(w)$ and a fixed CTL-formula φ.

Before we begin with the details of our reduction, let us discuss some problems that we need to overcome:

[3]NC^1 is the class of decision problems solvable by a uniform family of Boolean circuits, with polynomial size, depth $O(\lg(n))$, and fan-in 2.

Algorithm 3 Algorithm deciding $w \in L$ for an EXPSPACE-language L given via its serialisation $\mathcal{M}$, R and p.

```
Input: $w \in \{0,1\}^*$, $\mathcal{M}$, $\mathcal{A}_R = (S, s_0, F, \Delta)$, $p : \mathbb{N} \to \mathbb{N}$
  $s := s_0$
  $b := \epsilon;\ d := 0$
  $m := \exp^2(p(|w|))$
  while $d < m$ do
    $b := \mathrm{bin}_{(\lg m)}(d)$
    $s := \Delta(s, \chi_{\mathcal{M}}(w \cdot b))$
    $d := d + 1$
  end while
  return $s \in F$
```

(a) As Algorithm 3 stands, it requires exponential space in $|w|$ in order to store the values of b, d and m. This excludes the possibility of encoding those variables into the control locations of $\mathcal{A}_L(w)$. A possible solution to this problem is to store the values of the variables on the counter. However, when we want to compute b, we need to access the bit representation of d, and the binary representation of d comprises of exponentially many bits. Moreover, we need to be able to check if $d = m$, which cannot be done by simply subtracting m from the counter and performing a zero-test, since the size of m is too large.

(b) In [50], the problem of accessing the bit representation of d is solved by storing d in Chinese remainder representation using the first $\lg m$ prime numbers. Each such prime number can be represented in $O((p(|w|)^2)$ bits. However, we need exponentially many of them, and for that reason we cannot hard-wire them into $\mathcal{A}_L(w)$.

(c) When we want to compute $\chi_{\mathcal{M}}(w \cdot b)$, we need to simulate $\mathcal{M}$ on an input exponential in $|w|$. A pointer to the input $w \cdot b$ can be stored using $O(p(|w|))$ bits. However, we need to provide some on-the-fly mechanism to extract the bit that such a pointer points to.

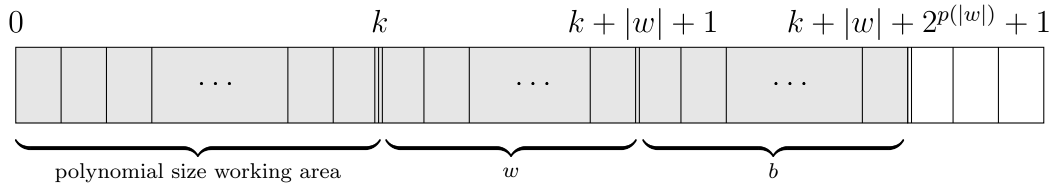

Figure 5.3: Segmentation of the counter used in the reduction. Here, $k \in \mathbb{N}$ is polynomial in $|w|$. Switching to the consecutive b, *i.e.*, incrementing d in Algorithm 3, can be simulated by adding $\exp(k + |w| + 1)$ to the counter.

There are a number of key insights that allow us to overcome those problems. The first is that the branching provided by CTL allows us to test the value of a bit of the counter without destroying the counter value, provided the address of the bit we test is polynomial in $|w|$. This enables us to toggle individual bits of the counter without affecting the other bits. We exploit this insight in order to partition the counter into various segments that can be used to store information that can individually be accessed. The second insight is that we can simulate in these segments computations of space-bounded Turing machines that require polynomial space in $|w|$ for their computations. To this end, we reserve a segment of the counter that is going to serve as the working tape of such a Turing machine. This is, for example, going to enable us to run $\mathcal{M}$ and to compute prime numbers on-the-fly. Third, we show that it is possible to compute the residue class of the counter modulo the i-th prime p_i, where i is exponential in $|w|$. This will enable us to test a bit of the counter whose address is exponential in $|w|$ by computing the Chinese remainder representation of the current counter value on-the-fly. All those insights are eventually going to enable us to simulate Algorithm 3.

We are now going to proceed with the technical details of our hardness proof. Let us consider Figure 5.3. It shows a bit representation of a counter value, where the least-significant bit is on the left-hand side and the most-significant bit is on the right-hand side. As discussed above, we aim for partitioning the counter into segments with some intended purpose. In our hardness proof, for some k polynomial in $|w|$ to be determined later, given a counter value $n \in \mathbb{N}$, $n[k, k + |w| + 1]$ is supposed to encode w. Moreover, $n[k + |w| + 1, k + |w| + \exp(p(|w|)) + 1]$ is supposed to encode b from Algorithm 3. Thus, $n[k, k + |w| + p(|w|) + 1] = w \cdot b$ and if we could find a way to evaluate $\chi_{\mathcal{M}}(w \cdot b)$, we could simulate one cycle of the **while**-loop of Algorithm 3. Simulating the consecutive cycle of this loop would then be possible by adding $\exp(k + |w| + 1)$ to the counter. In our reduction, we are going to use the segment $n[0, k]$ of the counter as storage space for the working tape of $\mathcal{M}$ and as some further temporary storage.

We are now going to provide a number queries that we use in order to simulate Algorithm 3 and which we implement via a number of model-checking problems of one-counter automata gadgets and CTL formulae. All gadgets are computable in polynomial time, and we will omit mentioning this fact in order to improve readability. The queries form a hierarchy and build on top of each other, and the last query establishes our hardness proof. The first two queries deal with extracting information encoded into the counter that enable us to then construct more sophisticated queries. In the following, let $n \in \mathbb{N}$ be a counter value, and let $i \in \mathbb{N}$ be a number given in unary, $I = (n[i_b, i_e])_2$ for $i_b, i_e \in \mathbb{N}$, $J = (n[j_b, j_e])_2$ for $j_b, j_e \in \mathbb{N}$ and $K = (n[k_b, \infty])_2$ for $k_b \in \mathbb{N}$ given in unary.

(i) What is the value of $\mathrm{bit}_i(n)$?

(ii) Is $K \equiv J \mod I$, provided $J \in [0, I - 1]$?

(iii) Is J the I-th prime number?

(iv) What is the value of $\mathrm{bit}_I(n)$?

(v) Is $n[k, k + |w| + p(|w|) + 1] \in L(\mathcal{M})$?

(vi) Is $w \in L$?

We now show how to implement Query (i), which is realised with a gadget $\mathcal{A}_{bit}$.

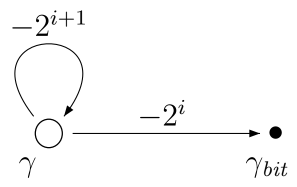

Figure 5.4: Gadget $\mathcal{A}_{bit}(i)$ used for testing the value of i-th bit of the counter.

Lemma 5.1.11 *For any $i \in \mathbb{N}$ given in unary, there exists a one-counter automaton $\mathcal{A}_{bit}(i)$ with a control location q and fixed CTL-formulae $\varphi_{bit,b}, b \in \{0,1\}$ such that for any $n \in \mathbb{N}$, $(T(\mathcal{A}_{bit}(i), (q,n)) \models \varphi_{bit,b}$ if, and only if, $b = \mathrm{bit}_i(n)$.*

Proof. Consider the automaton $\mathcal{A}_{bit}(i)$ in Figure 5.4 and let q be the ◯ location. Define the required CTL-formulae as

$$\varphi_{bit,0} \overset{\text{def}}{=} \gamma \wedge \mathsf{EF}(\gamma \wedge \neg\mathsf{EX}\gamma \wedge \neg\mathsf{EX}\gamma_{bit})$$
$$\varphi_{bit,1} \overset{\text{def}}{=} \gamma \wedge \mathsf{EF}(\gamma \wedge \neg\mathsf{EX}\gamma \wedge \mathsf{EX}\gamma_{bit}),$$

which can easily be seen to have the desired property. □

We continue with the gadget $\mathcal{A}_{mod}$, which realises Query (ii) and uses $\mathcal{A}_{bit}$ as an oracle.

Lemma 5.1.12 *Let $i_b < i_e < j_b < j_e < k \in \mathbb{N}$ be given in unary. For any $n \in \mathbb{N}$, let $I = (n[i_b, i_e])_2, J = (n[j_b, j_e])_2$ and $K = (n[k, \infty])_2$. Assuming $J \in [0, I-1]$, there exist a one-counter automaton $\mathcal{A}_{mod}(i_b, i_e, j_b, j_e, k)$ with a control location q and a fixed CTL-formula φ_{mod} such that $(T(\mathcal{A}_{mod}), (q,n))$ if, and only if, $K \equiv J \mod I$.*

Proof. Consider the one-counter automaton $\mathcal{A}_{mod}(i_b, i_e, j_b, j_e, k)$ in Figure 5.5, where q is the ◯ location. $\mathcal{A}_{mod}$ consists of two rows, each consisting of $i_e - i_b$ respectively $j_e - j_b$ diamonds. We aim for achieving that any time we traverse the upper row we subtract I from K. Likewise, the lower row is supposed to subtract J from K. Thus, if there is a path reaching the location labelled with γ_2 such that it is not possible to reach the location labelled with γ_{mod}, we have $K \equiv J \mod I$. In both rows row, each diamond is connected to an $\mathcal{A}_{bit}$ gadget. The CTL formula we are

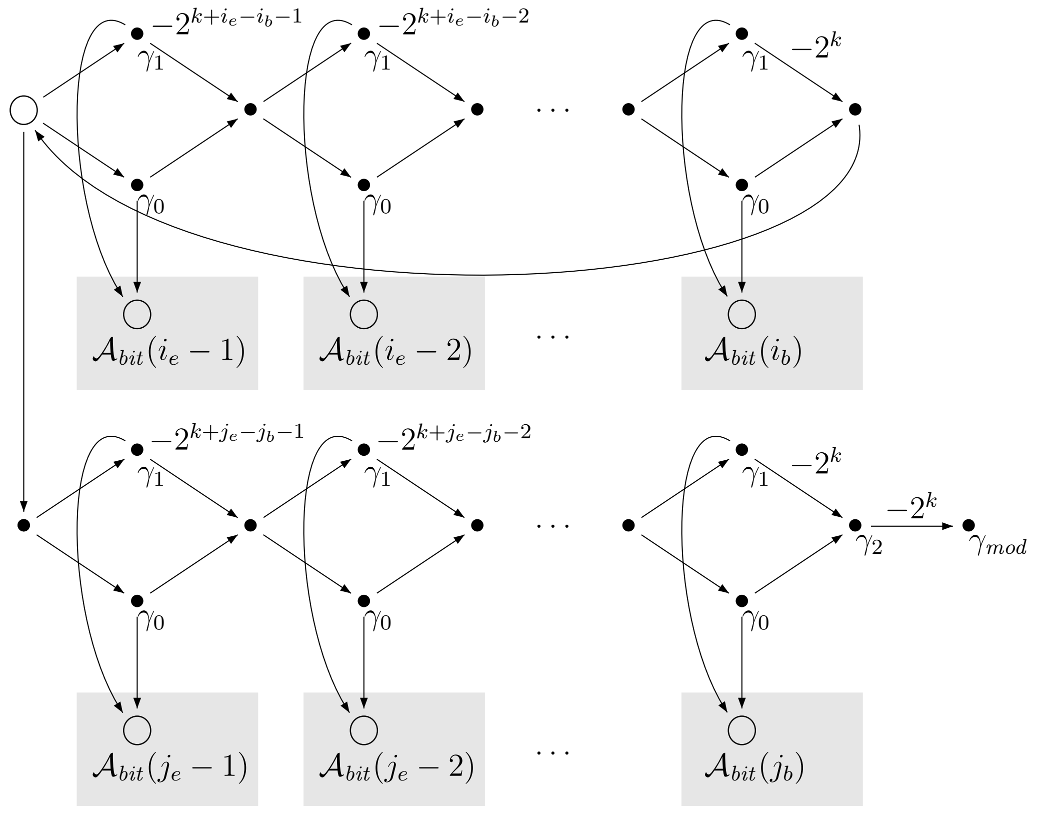

Figure 5.5: The one-counter automaton $\mathcal{A}_{mod}(i_b, i_e, j_b, j_e, k)$ used for the implementation of Query (ii).

going to define below will make sure that any time we traverse a diamond through a location labelled with γ_1, $\mathsf{EX}\varphi_{bit,1}$ holds, where $\varphi_{bit,1}$ is defined as in Lemma 5.1.11. For example, in the first diamond in the upper row this will ensure that we only subtract $\exp(k + i_e - i_b - 1)$ from the counter if, and only if, $\mathrm{bit}_{i_e-1}(n) = 1$ for the current counter value n. This allows us to construct a suitable CTL-formula that ensures that the gadgets $\mathcal{A}_{bit}$ can be used to "guide" the paths through $\mathcal{A}_{mod}$. We set

$$\varphi_{mod} \overset{\text{def}}{=} \mathsf{E}\left((\gamma_0 \to \mathsf{EX}\varphi_{bit,0} \wedge \gamma_1 \to \mathsf{EX}\varphi_{bit,1})\mathsf{U}(\gamma_2 \wedge \neg\mathsf{EX}\gamma_{mod})\right),$$

which can be seen to have the desired properties. □

We now turn towards Query (iii). Instead of giving a direct implementation, we sketch how an arbitrary space-bounded Turing machine can be simulated via an instance of a model checking problem. A concrete implementation of Query (iii) then follows as an instantiation of the next lemma with a Turing machine $\mathcal{M}$ that computes prime numbers.

Lemma 5.1.13 *Let $i_b < i_e < j_b < j_e < k_b < k_e \in \mathbb{N}$ be given in unary and let $\mathcal{M}$ be a space-bounded deterministic Turing machine such that $\mathcal{M}$ uses at most $j_e - j_b$ tape cells on an input of size $i_e - i_b$. There exists a one-counter automaton $\mathcal{A}_{\mathcal{M}}(i_b, i_e, j_b, j_e, k_b, k_e)$ with a control location q and a fixed CTL-formula $\varphi_{\mathcal{M}}$ such that for all $n \in \mathbb{N}$, $(T(\mathcal{A}_{\mathcal{M}}), (q, n)) \models \varphi_{\mathcal{M}}$ if, and only if, $\mathcal{M}$ has on input $n[i_b, i_e]$ a run that ends in an accepting state in which the content of the working tape is $n[k_b, k_e]$.*

Proof. The idea is to simulate a run of $\mathcal{M}$ on input $n[i_b, i_e]$ using the cells in $n[j_b, j_e]$ as the working tape of $\mathcal{M}$ until we reach an accepting state of $\mathcal{M}$. Once an accepting state has been reached, we can compare the contents of $n[j_b, j_e]$ and $n[k_b, k_e]$ with an additional gadget. We omit details of this additional gadget for brevity and concentrate on the simulation of $\mathcal{M}$. The construction of such a gadget is an easy exercise and can be realised using the gadget $\mathcal{A}_{bit}$ constructed in the implementation of Query (i).

The one-counter automaton $\mathcal{A}_{\mathcal{M}}$ contains a gadget $\mathcal{A}_{bit}(m)$ for each $m \in [i_b, i_e - 1] \cup [j_b, j_e - 1]$ and additional control locations

$$S \times ([0, i_e - i_b + 1]) \times ([0, j_e - j_b + 1]) \times \{0,1\} \times \{0,1\}$$

that we use to simulate runs of $\mathcal{M}$. The intention behind those control locations is as follows: a tuple (s, i, j, b_1, b_2) corresponds to the configuration of $\mathcal{M}$ in which $\mathcal{M}$ is in control state s, the input head of $\mathcal{M}$ is at position i reading b_1 and the working head of $\mathcal{M}$ is at position j reading b_2. We will use $i = 0$ and $j = 0$ to indicate that the input respectively working head has reached the left delimiter $\triangleright$, and $i_e - i_b + 1$ to indicate that the input head has reached the right delimiter $\triangleleft$. The content of the whole input or working tape is *not* encoded in the control locations of $\mathcal{M}$, but in the respective segments of the counter, which keeps the number of control locations of $\mathcal{A}_{\mathcal{M}}$ polynomial.

Let us explain how $\mathcal{A}_{\mathcal{M}}$ is wired and how it works with the help of an example shown in Figure 5.6. Here, we assume that $\mathcal{A}_{\mathcal{M}}$ is in control location $(s, i, j, 0, 1)$,

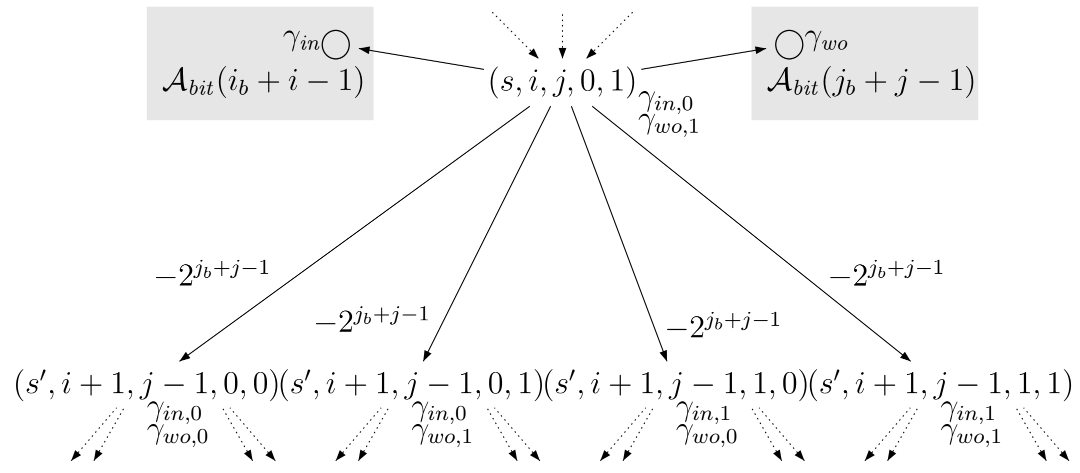

Figure 5.6: Part of the one-counter automaton $\mathcal{A}_{\mathcal{M}}(i_b, i_e, j_b, j_e)$ for the case when $\mathcal{M}$ is in state s and the input head scans cell i reading 0, the working head of $\mathcal{M}$ scans cell j reading 1 and the transition function requires the input head to move to the right, the output head to the left and to write a 0 to the current working tape cell.

$i \in [i_e - i_b]$, $j \in [j_e - j_b]$ and that the transition relation Δ of $\mathcal{M}$ is such that $\Delta(s, 0, 1) = (s', 1, -1, 0)$. Thus, we simulate a transition in which $\mathcal{M}$ is in control state s, reading a 0 on the input tape, 1 on the working tape, and the transition function requires the input tape head to move to the right, the output tape head to move to the left, to write a 0 on the current working tape position and to switch to control state s'. Each (s, i, j, b_1, b_2) of $\mathcal{A}_{\mathcal{M}}$ is labelled with atomic propositions γ_{in,b_1} and γ_{wo,b_2} and connects to a gadget $\mathcal{A}_{bit}(i_b+i-1)$ and $\mathcal{A}_{bit}(j_b+j-1)$. Consequently, $(s, i, j, 0, 1)$ in Figure 5.6 is labelled with $\gamma_{in,0}$ and $\gamma_{wo,1}$. Whenever we reach a control location (s, i, j, b_1, b_2), this labelling allows us to verify that the bits at the respective positions of the counter actually correspond to the intended content of the tapes of $\mathcal{M}$. Now $(s, i, j, 0, 1)$ has a transition to each $(s', i+1, j-1, b_1', b_2'), b_1', b_2' \in \{0, 1\}$, and along each transition we subtract $\exp(j_b + j - 1)$ from the counter. The four transitions allow us to guess the content of the input respectively working tape at the updated head positions. The validity of our guess can be verified in the next step using the gadgets $\mathcal{A}_{bit}$. Subtracting $\exp(j_b + j - 1)$ from the counter simulates writing 0 at the current position of the working head. Extra effort is needed to get the behaviour for control locations of the form (s', i', j', b'_1, b'_2) with $i' \in \{0, i_e - i_b + 1\}$ or $j' \in \{0, j_e - j_b + 1\}$ right, *i.e.*, when $\mathcal{M}$ has reached a delimiter. However, this is a rather technical than conceptual issue and will not be considered here. Finally any (s, i, j, b_1, b_2) such that $s \in A$, *i.e.*, s is an accepting location, is labelled with a proposition γ_A and connected to a gadget that allows for testing whether the content of $n[j_b, j_e]$ is the same as $n[k_b, k_e]$ via a CTL formula φ_{eq} that holds if, and only if, $n[j_b, j_e] = n[k_b, k_e]$. Let $\varphi_{bit,0}, \varphi_{bit,1}$ be the CTL-formulae from Lemma 5.1.11. We define the CTL formula required in the lemma as

$$\varphi_{\mathcal{M}} \overset{\text{def}}{=} \mathsf{E}\Big(\bigwedge_{b_1, b_2 \in \{0,1\}} (\gamma_{in,b_1} \to \mathsf{EX}(\gamma_{in} \wedge \varphi_{bit,b_1}) \wedge \gamma_{wo,b_2} \to \mathsf{EX}(\gamma_{wo} \wedge \varphi_{bit,b_2}))\mathsf{U}(\gamma_A \wedge \mathsf{EX}\varphi_{eq})\Big).$$

Thus, when adding a distinguished control location q to $\mathcal{A}_{\mathcal{M}}$ that non-deterministically branches into control locations $(s_0, 1, 1, b_1, b_2), b_1, b_2 \in \{0, 1\}$, we have that for all $n \in \mathbb{N}$, $(T(\mathcal{A}), (q_{\mathcal{M}}, n)) \models \varphi_{\mathcal{M}}$ if, and only if, $\mathcal{M}$ has a run on input $n[i_b, i_e]$ that ends in an accepting state in which the content of the working tape is $n[k_b, k_e]$. □

We are now going to consider an implementation of Query (iv), which is about extracting bits from the counter whose address is encoded in binary into the counter. With Figure 5.3 in mind, we will be interested in testing bits in the upper segment of the counter beyond the working area. The main challenge we need to overcome is that we cannot adopt the idea from Query (i) and use loops in a counter automaton to test for divisibility, as the address of the bit we test for can be exponential. Instead, we represent the counter value in Chinese remainder representation and then use Bit-From-Crr in order to extract the desired bit.

Lemma 5.1.14 *Given $n \in \mathbb{N}$ and $i_b < i_e < k \in \mathbb{N}$ in unary, and let $i = (n[i_b, i_e])_2$ and $m = (n[k, \infty])$. Provided k is sufficiently large, there exists a one-counter automaton $\mathcal{A}_{binbit}(i_b, i_e, k)$ with a control location q and a fixed CTL formula $\varphi_{binbit,b}, b \in \{0, 1\}$ such that $(T(\mathcal{A}_{binbit,b}), (q, n)) \models \varphi_{binbit,b}$ if, and only if,* $\text{bit}_i(m) = b$.

Proof. As discussed above, we determine $\text{bit}_i(m)$ through the Chinese remainder representation of m. By Proposition 5.1.5, there is an L-Turing machine $\mathcal{M}$ computing

$\mathrm{bit}_i(m)$. Since i is given in binary, $\mathcal{M}$ can compute $\mathrm{bit}_i(m)$ in space polynomial in $i_e - i_b$. Simulating $\mathcal{M}$ can be done in a similar way as described in Lemma 5.1.13. However, the input $\mathrm{CRR}_s(m)$ with $s = \exp(i_e - i_b)$ cannot be encoded in the counter as it is exponential in $i_e - i_b$. Instead, we sketch below how the construction in Lemma 5.1.13 can be altered in a way such that the input to $\mathcal{M}$ is computed on-the-fly.

Recall that $\mathrm{CRR}_s(m) = (b_{j,0} \ldots b_{j,p_j-1})_{j\in[s]}$, where p_j is the j-th prime number. In order to compute a fixed $b_{j,r}$, $\mathcal{A}_{binbit}$ uses a segment of the counter distinct from $[i_b, i_e]$ and $[k, \infty]$ in which it stores the index $j \in [s]$, p_j and $r \in [0, p_j - 1]$ in binary, which serve as pointers to the Chinese remainder representation of m. We can then employ $\mathcal{A}_{mod}$ in order to test whether $m \equiv r \mod p_j$, *i.e.*, compute $b_{j,r}$. In order to compute the j-th prime number p_j, $\mathcal{A}_{binbit}$ employs a one-counter automaton $\mathcal{A}_{\mathcal{M}'}$ that we obtain from Lemma 5.1.13 whose working tape is stored in some unused segment of the counter of size polynomial in $\lg s$. Consequently, when simulating $\mathcal{M}$ as in Lemma 5.1.13 and guessing the current input symbol, $\mathcal{A}_{binbit}$ uses $\mathcal{A}_{mod}$ in order to verify the guess. Simulating a movement of the input head of $\mathcal{M}$ is done as follows. If $r \in [1, p_j - 2]$ then simulating that the input head moves to the left or right corresponds to decrementing respectively incrementing r. If $r = 0$ and we wish to simulate that the input head moves to the left then the index j is decremented, p_j is re-computed and r is set to $p_j - 1$. The case when $r = p_j - 1$ and we wish to simulate that the input head moves to the right follows analogously.

As discussed above, storing the working tapes of $\mathcal{M}$, $\mathcal{M}'$ and the additional storage requires segments of size polynomial in $i_e - i_b$. Hence, if k is sufficiently, polynomially large, those segments can be reserved in the working area of the counter and the lemma follows. $\square$

Using $\mathcal{A}_{binbit}$ as a gadget, we can now sketch an implementation of Query (v). This query requires us to simulate the computation of a logarithmically-space bounded Turing machine on an exponentially large input which is encoded into a segment of the counter.

Lemma 5.1.15 *Given an* L*-Turing machine* $\mathcal{M}$*,* $n \in \mathbb{N}$*,* $k \in \mathbb{N}$ *in unary and* $l \in \mathbb{N}$

in binary, and let $m = n[k, k + l]$. Provided k is sufficiently large, there exists a one-counter automaton $\mathcal{A}_{\mathcal{M}}(k, l)$ and a fixed CTL *formula $\varphi_{\mathcal{M}}$ such that $(T(\mathcal{A}), (q, n)) \models \varphi_{\mathcal{M}}$ if, and only if, $\mathcal{M}$ accepts input m.*

Proof. We sketch a proof which combines the ideas from the Lemmas 5.1.13 and 5.1.14. We cannot directly apply Lemma 5.1.13, since the input to $\mathcal{M}$ is of exponential length. Instead, we will extract the input to $\mathcal{M}$ bit by bit using the gadget $\mathcal{A}_{binbit}$ from Lemma 5.1.14.

Since the input to $\mathcal{M}$ is of exponential length, $\mathcal{M}$ runs in PSPACE and we can reserve a segment in the working area of the counter below k which stores the working tape during the simulation of $\mathcal{M}$. Moreover, we are going to use an additional segment $n[j_b, j_e]$ below k in order to store a pointer to the input m. This segment requires a linear number of bits in the size of l.

The simulation of $\mathcal{M}$ can be done in a similar way as in Lemma 5.1.13. In particular, the symbol read on the input tape is guessed when moving the head, however, as in Lemma 5.1.14, we cannot use φ_{bit} from Lemma 5.1.11 in order to validate our guess. Instead, we use $\mathcal{A}_{binbit}(j_b, j_e, k)$ and φ_{binbit} from Lemma 5.1.14 for this purpose. Simulating moving the input head can be done by incrementing respectively decrementing the pointer in $n[j_b, j_e]$. $\mathcal{A}_{binbit}(j_b, j_e, k)$ requires some additional segments in the working area of the counter, but only of size polynomial in the size of l. Consequently, if k is chosen sufficiently large, polynomial in l, $\mathcal{M}$ can be simulated on input m via $\mathcal{A}_{\mathcal{M}}$ and a CTL formula $\varphi_{\mathcal{M}}$. □

Finally, we can turn towards an implementation of Query (vi), which builds on top of all previously defined gadgets concludes our proof of EXPSPACE-hardness of CTL model checking of one-counter automata.

Lemma 5.1.16 *Let $L \subseteq \{0, 1\}^*$ be a language in* EXPSPACE *and $w \in \{0, 1\}^*$. There exists a one-counter automaton $\mathcal{A}_L(w)$ with a control location q, $n \in \mathbb{N}$ and a fixed* CTL*-formula φ such that $(T(\mathcal{A}), (q, n)) \models \varphi$ if, and only if, $w \in L$.*

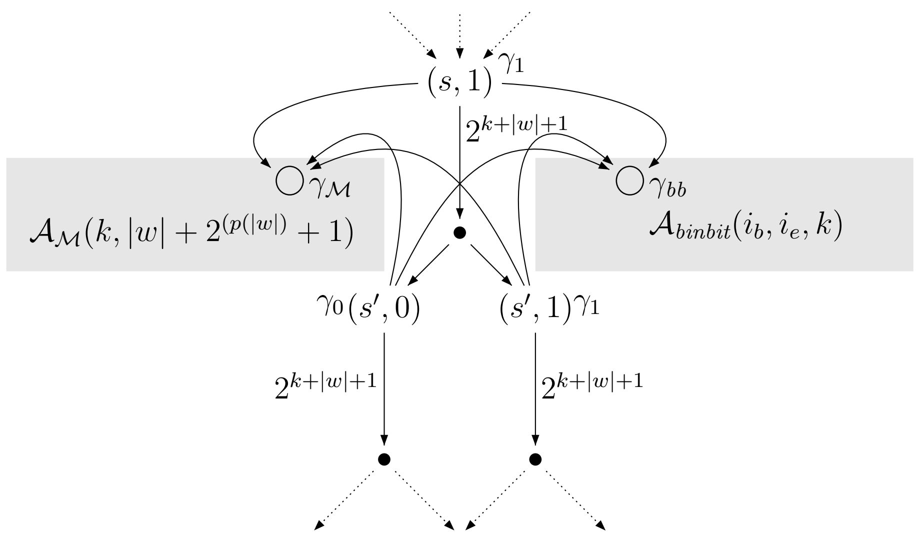

Figure 5.7: Part of the one-counter automaton $\mathcal{A}_L$ constructed in Lemma 5.1.16.

Proof. By Theorem 2.4.2 there are an L-Turing machine $\mathcal{M}$ and a polynomial p such that for any $w \in \{0,1\}^*$ and $m = \exp^2(p(|w|))$,

$$w \in L \Leftrightarrow \left(\chi_{\mathcal{M}}(w \cdot \mathrm{bin}_{(\lg m)}(d))\right)_{d \in [0,m-1]} \in R.$$

Let $\mathcal{A}_R = (S, s_0, F, \Delta)$ be a deterministic finite-state automaton defining R. We are now going to sketch the construction of a one-counter automaton $\mathcal{A}_L$ that simulates Algorithm 3. To this end, as discussed at the beginning of the hardness-proof, we partition the counter into segments as sketched in Figure 5.3. Hence, for a given counter value n and some $k \in \mathbb{N}$ to be determined later, $n[k, k+|w|]$ stores w and $n[k+|w|, k+|w|+\exp(p(|w|))+1]$ stores a bit-string of length $\exp(p(|w|))$ which represents b from Algorithm 3.

In order to compute $\chi_{\mathcal{M}}(w \cdot b)$, $\mathcal{A}_L$ contains a gadget $\mathcal{A}_{\mathcal{M}}(k, |w|+\exp(p(|w|))+1)$ as defined in Lemma 5.1.15. Furthermore, $\mathcal{A}_L$ contains a gadget $\mathcal{A}_{binbit}(i_b, i_e, k)$ from Lemma 5.1.14 for some $i_b < i_e < k$ that we are going to use in order to test whether $\mathrm{bit}_{k+|w|+\exp(p(|w|))+1}(n) = 1$ for a given counter value n. This is going to enable us to determine when to quit from the **while**-loop in Algorithm 3.

We are now going to concentrate on the simulation of the body of the **while**-loop in Algorithm 3. In order to simulate a run of $\mathcal{A}_R$, $\mathcal{A}_L$ contains two control locations

$(s,0)$ and $(s,1)$ for each $s \in S$ that indicate that $\mathcal{A}_R$ is simulated to be in state s reading 0 respectively 1. A control location $(s,0)$ is labelled with γ_0 and $(s,1)$ with γ_1. Moreover, each $s \in F$ is labelled with γ_F in order to indicate that an accepting state has been reached. The transitions between the $(s,c), c \in \{0,1\}$ control locations of $\mathcal{A}_L$ are sketched in Figure 5.7. Each (s,c) has a transition to $\Delta(s,c)$, which adds $\exp(k+|w|+1)$ to the counter, thus simulating an increment of d. Moreover, each (s,c) has a transition to the gadget $\mathcal{A}_{\mathcal{M}}$ and $\mathcal{A}_{binbit}(i_b, i_e, k)$. The initial locations of the gadgets are labelled with $\gamma_{\mathcal{M}}$ and γ_{bb}, respectively. The transition to $\mathcal{A}_{\mathcal{M}}$ enables us to verify that a guessed value of $\chi_{\mathcal{M}}(w \cdot b)$ is actually correct. The other transition to $\mathcal{A}_{binbit}(i_b, i_e, k)$ allows us to check for a counter value n if $\mathrm{bit}_{k+|w|+\exp(p(|w|))+1}(n) = 1$.

We are now going to give the required CTL formula φ, which is defined as follows:

$$\varphi \overset{\text{def}}{=} \mathsf{E}(\gamma_0 \to \mathsf{EX}(\gamma_{\mathcal{M}} \wedge \neg\varphi_{\mathcal{M}}) \wedge \gamma_1 \to \mathsf{EX}(\gamma_{\mathcal{M}} \wedge \neg\varphi_{\mathcal{M}}) \wedge \neg(\mathsf{EX}(\gamma_{bb} \wedge \varphi_{binbit}))\mathsf{U}$$
$$(\gamma_F \wedge \mathsf{EX}(\gamma_{bb} \wedge \varphi_{binbit})).$$

Here, φ_{binbit} and $\varphi_{\mathcal{M}}$ are the CTL formulae from the Lemmas 5.1.14 and 5.1.15. Informally speaking, φ makes sure that we guess the value of $\chi_{\mathcal{M}}(w \cdot b)$ correctly as long as $d < \exp^2(p(|w|))$. Once $d = \exp^2(p(|w|))$, φ requires that the simulation of $\mathcal{A}_R$ ends in an accepting state.

It remains to discuss the size of k. The value of k needs to be chosen sufficiently large such that the gadget $\mathcal{A}_{\mathcal{M}}$ can work properly. By a similar argument as in the discussion at the end of the proof of Lemma 5.1.15, $\mathcal{A}_{\mathcal{M}}$ requires k to be polynomial in $|w|$. Moreover, the gadget $\mathcal{A}_{binbit}$ also requires k to be polynomial in $|w|$, as i_b and i_e only need to be polynomial in $|w|$.

The counter value n required in the lemma is the unique natural number $n \in \mathbb{N}$ such that $n[k, k+|w|] = w$, $(n[i_b, i_e])_2 = \exp^2(p(|w|))$, and all other bits of n are zero. Clearly, this n can be computed in polynomial time and, informally speaking, provides a proper initial configuration of $\mathcal{A}_L$.

Finally, by introducing a distinguished control location q to the control locations of $\mathcal{A}_L$ that connects to $(s_0, 0)$ and $(s_0, 1)$, we have $(T(\mathcal{A}_L(w)), (q, n)) \models \varphi$ if, and only if, $w \in L$. $\square$

Together with Proposition 5.1.1, we can now deduce the main theorem of this section.

Theorem 5.1.3 CTL *model checking of one-counter automata is* $\mathsf{EXPSPACE}$*-complete already for a fixed* CTL *formula.*

CTL Model Checking of Parametric-One Counter Automata

Since EF is a syntactic fragment of CTL, it follows from the results in Section 5.1.1 that model checking CTL formulae on parametric one-counter automata is Π^0_1-hard. In this section, we strengthen this result for the CTL case and show that Π^0_1-hardness can already be achieved for a fixed CTL formula and a parametric one-counter automaton with only one parameter.

We reduce from the reachability problem for two-counter automata. Given a two-counter automaton $\mathcal{A}'$ and two locations q, q' of $\mathcal{A}'$, we construct a parametric one-counter automaton $\mathcal{A}$ with one parameter y from $\mathcal{A}'$ such that $(q, \vec{0}) \to^*_{\mathcal{A}'} (q', \vec{0})$ if, and only if, $(T(\mathcal{A}), (q, 0)) \not\models \varphi$. For our purposes, we may assume with no loss of generality that counter updates of $\mathcal{A}'$ are in unary, *i.e.*, of the form $\mathsf{add}_i(z)$ for $i \in \{0, 1\}$ and $z \in \{-1, 0, +1\}$. Moreover, we assume that the first and second counter of $\mathcal{A}'$ are tested for zero before q' can be reached. As in the hardness proof in Section 4.2, we exploit the fact that on a witnessing run there exists an $m \in \mathbb{N}$ such that none of the two counters of $\mathcal{A}'$ exceeds this value. We use the parameter y in order to guess m, which allows us to give a one-to-one correspondence between configurations of $\mathcal{A}'$ and $\mathcal{A}$. Given a configuration (q, n_1, n_2) of $\mathcal{A}'$ with $n_1, n_2 < m$ and a valuation such that $\nu(y) = m$, the corresponding configuration of $\mathcal{A}$ is (q, n), where $n = mn_2 + n_1$, *i.e.*, $n_1 \equiv n \mod m$ and $n_2 = n \operatorname{div} m$. Testing the first and the second counter of $\mathcal{A}'$ for zero corresponds to checking whether whether $n \equiv 0 \mod m$ respectively $n < m$. In our reduction, we use the branching that CTL formulae offer in order to perform these tests without destroying the value of the counter. Incrementing and decrementing the first counter of $\mathcal{A}'$ can be simulated by adding respectively subtracting 1 from the counter of $\mathcal{A}$. Regarding the second counter of $\mathcal{A}'$, incrementation and decrementation correspond to adding respectively subtracting m, *i.e.*, the value of the parameter y,

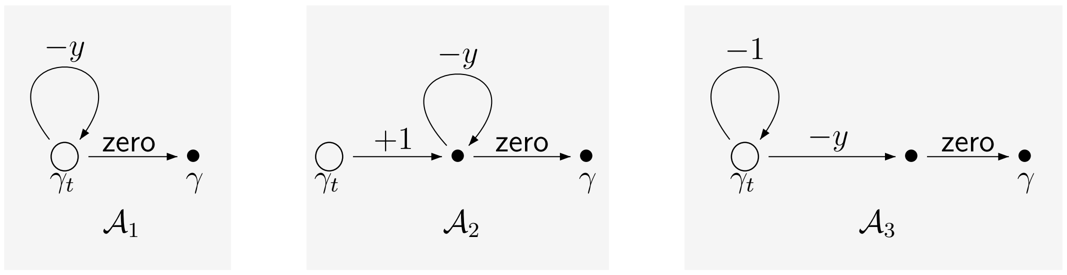

Figure 5.8: Gadgets used in the reduction from two-counter automata reachability to CTL model checking of a parametric one-counter automaton.

from the counter. Of course, we have to ensure that we do not overflow when we perform these operations. For example, suppose that $n \equiv -1 \mod m$. If we intend to add 1 to the counter of $\mathcal{A}$ in order simulate an increment of the first counter of $\mathcal{A}'$, informally speaking we would accidentally reset the first counter of $\mathcal{A}'$ and increment its second counter. Again, we will use the branching that CTL formulae offer in order to make increments and decrements safe.

We begin the formal part of the reduction by providing some gadgets that allow us to perform the necessary tests described above.

Lemma 5.1.17 *There exist fixed parametric one-counter automata $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ with one parameter y, each with a control location q, and a fixed* CTL *formula φ_t such that for all valuations ν and $n \in \mathbb{N}$*

(i) $(T(\mathcal{A}_1^\nu), (q, n)) \models \varphi_t$ *if, and only if,* $n \not\equiv 0 \mod \nu(y)$*;*

(ii) $(T(\mathcal{A}_2^\nu), (q, n)) \models \varphi_t$ *if, and only if,* $n \not\equiv -1 \mod \nu(y)$*; and*

(iii) $(T(\mathcal{A}_3^\nu), (q, n)) \models \varphi_t$ *if, and only if,* $n < \nu(y)$*.*

Proof. The parametric one-counter automata $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ are depicted in Figure 5.8. It is now easily verified that $\varphi_t \overset{\text{def}}{=} \neg(\mathsf{EF}\gamma)$ when evaluated in the locations labelled with ◯ is a CTL formula with the desired properties. □

Figure 5.9 shows the replacement rules that we apply in order to obtain $\mathcal{A}$ from $\mathcal{A}'$. The top row deals with operations on the first counter: any transition acting on the

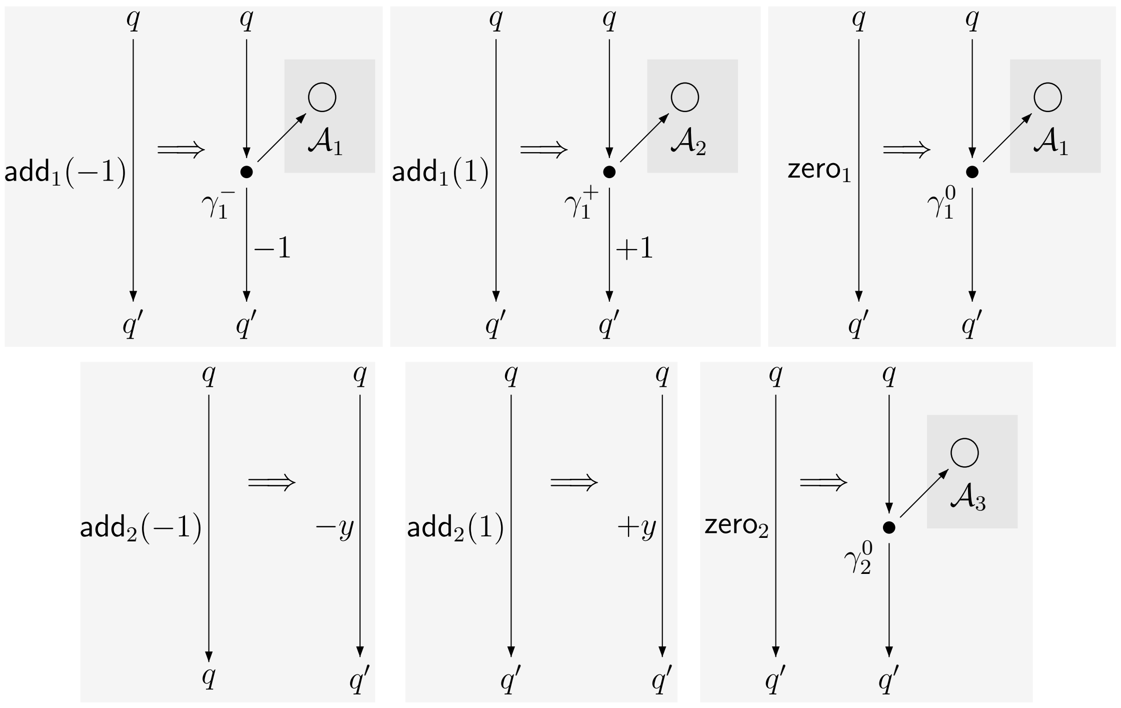

Figure 5.9: Replacement rules for obtaining a parametric one-counter automaton $\mathcal{A}$ from a given two-counter automaton $\mathcal{A}'$ used in the hardness proof of CTL model checking.

first counter of $\mathcal{A}'$ is split by introducing a fresh control location that is connected to an $\mathcal{A}_1$ or $\mathcal{A}_2$ as defined in Lemma 5.1.17 and labelled with an extra label depending on which operation is performed in $\mathcal{A}'$. This allows us to use the formula φ_t defined in Lemma 5.1.17 in order to check if the value of the counter is congruent 0 or -1 modulo the value of y before we add respectively subtract 1 from the counter. The bottom row deals with operations performed on the second counter of $\mathcal{A}'$. All $\mathsf{add}_2(\pm 1)$-operations are replaced with a corresponding $\mathsf{add}(\pm y)$-operation. In order to simulate testing the second counter for zero, we again split the edge connecting any q and q' by introducing a fresh control location which is connected to an $\mathcal{A}_3$ gadget as defined in Lemma 5.1.17. This allows us to use the formula φ_t from the lemma in order to check if the value of the counter is less than the value of y. Additionally, we label q' with a fresh label $\gamma_{q'}$ in $\mathcal{A}$ as a marker indicating that control location q' has been reached. We now define the CTL formula for our hardness proof as follows:

$$\varphi_c \overset{\text{def}}{=} \big((\gamma_1^- \vee \gamma_1^+ \vee \gamma_2^0) \rightarrow \mathsf{EX}(\gamma_t \wedge \varphi_t)\big) \wedge \big(\gamma_1^0 \rightarrow \mathsf{EX}(\gamma_t \wedge \neg\varphi_t)\big)$$
$$\varphi \overset{\text{def}}{=} \mathsf{E}(\varphi_c \mathsf{U} \gamma_{q'}).$$

Suppose there exists a valuation ν such that $m = \nu(y)$ and $(T(\mathcal{A}^\nu), (q, 0)) \models \varphi$. There exists a finite path in $T(\mathcal{A}^\nu)$ starting in $(q, 0)$ along which φ_c holds and which ends in $(q', 0)$. Since φ_c ensures that all corresponding zero-tests in $\mathcal{A}'$ are matched and that all updates to the counter of $\mathcal{A}^\nu$ respect the boundary m, this path yields a run of $\mathcal{A}'$ witnessing $(q, 0) \rightarrow^*_{\mathcal{A}} (q', 0)$ on which both counters do not exceed m. The converse direction follows analogously. This shows that model checking CTL-formulae of parametric one-counter automata is Π^0_1-hard.

Membership in Π^0_1 is rather trivial as in the EF case. Given a parametric one-counter automaton $\mathcal{A}$ with parameters $y_1, \ldots, y_n$, a CTL formula φ and (q, n), we can enumerate all possible valuations ν of the parameters and check whether or not $(T(\mathcal{A}^\nu), (q, n)) \models \varphi$, which by Proposition 5.1.1 is decidable.

Theorem 5.1.4 *Model checking* CTL*-formulae on parametric one-counter automata is* Π^0_1*-complete already for parametric one-counter automata with only one parameter and a fixed* CTL*-formula.*

$$\begin{aligned}
\tau \models \gamma &\iff \gamma \in \tau(0)\\
\tau \models \neg\varphi &\iff \tau \not\models \varphi\\
\tau \models \varphi_1 \wedge \varphi_2 &\iff \tau \models \varphi_1 \text{ and } \tau \models \varphi_2\\
\tau \models \mathsf{X}\varphi &\iff \tau^1 \models \varphi\\
\tau \models \varphi_1 \mathsf{U} \varphi_2 &\iff \text{there is } j \in \mathbb{N} \text{ such that } \tau^j \models \varphi_2 \text{ and for all } i \in [0, j-1], \tau^i \models \varphi_1
\end{aligned}$$

Table 5.2: Semantics of LTL.

5.2 Linear-Time Temporal Logic (LTL) Model Checking

This section establishes the computational complexity of model checking formulae of linear-time temporal logic (LTL) on transition systems generated by one-counter automata and families of transition systems generated by parametric one-counter automata. In contrast to CTL, we are going to show that the model checking problem is decidable in both cases, PSPACE-complete in the former and coNEXPTIME-complete in the latter case. We begin with some standard definitions.

Formulae of LTL are inductively defined according to the following grammar, where γ ranges over a set of labels Λ:

$$\varphi ::= \gamma \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi.$$

The standard Boolean abbreviations and true are defined in the same way as in CTL. The *finally modality* $\mathsf{F}\varphi$ is an abbreviation for $\mathsf{true}\mathsf{U}\varphi$ and the *globally modality* $\mathsf{G}\varphi$ abbreviates $\neg\mathsf{F}\neg\varphi$. The size $|\varphi|$ of an LTL formula φ is defined as the number of symbols required to write it down. The semantics of LTL is given in terms of traces $\tau : \mathbb{N} \to 2^\Lambda$ and shown in Table 5.2. The model checking problems that we consider are defined as follows:

LTL OCA Model Checking

INPUT: A one-counter automaton $\mathcal{A}$, a configuration (q, n) of $\mathcal{A}$ and an LTL formula φ.

QUESTION: Does $\tau \models \varphi$ for every infinite trace τ starting in (q, n)?

LTL POCA Model Checking

INPUT: A parametric one-counter automaton $\mathcal{A}$, a configuration (q, n) of $\mathcal{A}$ and an LTL formula φ.

QUESTION: Does $\tau \models \varphi$ for every valuation $\nu : Y \to \mathbb{N}$ and every infinite trace τ starting in (q, n) in $T(\mathcal{A}^\nu)$?

All of our upper bounds rely on the standard automata-theoretic approach to LTL model checking [105]. In the following, we give a brief account of this approach. Expositions on this topic can be found in the literature, see *e.g.* [3, 33]. The heart of the approach is the construction of a Büchi automaton $\mathcal{A}_\varphi$ from an LTL formula φ whose transition system consists of all traces satisfying φ, formally captured by the following theorem.

Proposition 5.2.1 ([105]) *Given an LTL formula φ, there exists a corresponding Büchi automaton $\mathcal{A}_\varphi$ with the initial control location q_φ such that $|\mathcal{A}_\varphi| = \exp(O(|\varphi|))$ and for all traces τ, $\tau \models \varphi$ if, and only if, there is a Büchi run ϱ in $T(\mathcal{A}_\varphi)$ starting in q_φ with trace τ.*

Thus, for a given one-counter automaton $\mathcal{A}$ and an LTL formula φ, checking $(T(\mathcal{A}), (q, n)) \not\models \varphi$ can be reduced to checking whether there is a common trace τ in $T(\mathcal{A})$ starting in (q, n) and $T(\mathcal{A}_{\neg\varphi})$ starting in $q_{\neg\varphi}$. Consequently, this reduces to checking whether there is a Büchi run in $T(\mathcal{A}) \times T(\mathcal{A}_{\neg\varphi})$ which traverses states with a component of the final locations of $\mathcal{A}_{\neg\varphi}$ infinitely often. In the following, we show how to construct a one-counter automaton that generates the transition system $T(\mathcal{A}) \times T(\mathcal{A}_{\neg\varphi})$, which then allows us to decide the existence of a Büchi path via our results obtained in Chapter 4.

Let $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda, \xi)$ be a one-counter automaton with $Q = F$ and let $\mathcal{A}_\varphi = (Q_\varphi, \Lambda, q_\varphi, F_\varphi, \Delta_\varphi)$ be the Büchi automaton corresponding an LTL formula φ. We define the *product automaton* $\mathcal{A}'$ *of* $\mathcal{A}$ *and* $\mathcal{A}_\varphi$ as $\mathcal{A}' = (Q', \Lambda, q_0', F', \Delta', \lambda', \xi')$, where

- $Q' \overset{\text{def}}{=} \{(q, q_\varphi) \in Q \times Q_\varphi : \lambda(q) = \lambda_\varphi(q_\varphi)\}$;
- $F' \overset{\text{def}}{=} Q \times F_\varphi$;
- $\Delta' \overset{\text{def}}{=} \{((q, q_\varphi), (q', q_\varphi')) \in Q' \times Q' : (q, q') \in \Delta \text{ and } (q_\varphi, q_\varphi') \in \Delta_\varphi\}$;
- $\lambda'(q, q_\varphi) \overset{\text{def}}{=} \lambda(q)$; and
- $\xi'((q, q_\varphi), (q', q_\varphi')) \overset{\text{def}}{=} \xi(q, q')$.

We write $\mathcal{A} \times \mathcal{A}_\varphi$ to denote the product automaton of $\mathcal{A}$ and $\mathcal{A}_\varphi$. It is easily checked that $\mathcal{A} \times \mathcal{A}_\varphi$ is computable in L and that $T(\mathcal{A} \times \mathcal{A}_\varphi)$ is isomorph to $T(\mathcal{A}) \times T(\mathcal{A}_\varphi)$.

LTL Model Checking of One-Counter Automata

We are now going to show that the combined complexity of LTL model checking of one-counter automata is PSPACE-complete and coNP-complete if we fix φ. PSPACE-hardness of the model checking problem follows immediately from PSPACE-hardness of LTL model checking on Büchi automata [101]. In order to show membership in PSPACE, we employ the automata-theoretic approach discussed in the previous section. However, given a one-counter automaton $\mathcal{A}$ and an LTL formula φ, it is not sufficient to naïvely construct the product automaton $\mathcal{A} \times \mathcal{A}_{\neg\varphi}$ and then check for the existence of a Büchi path, since $|\mathcal{A} \times \mathcal{A}_{\neg\varphi}| = |\mathcal{A}| \exp(O(|\varphi|))$, which only gives a $\mathsf{coNEXPTIME}$ upper bound for LTL model checking. Instead, we reduce the model checking problem to a model checking problem in a *unary* one-counter automaton, similar to the CTL case.

Let $\mathcal{A}'$ be the unary one-counter automaton obtained from $\mathcal{A}$ by expanding transitions that increment the counter as described at the beginning of this chapter. As in the CTL case, in order to decide a model checking problem on $\mathcal{A}$ via $\mathcal{A}'$, we have to make some adjustments to the LTL formula φ. To this end, we define $\varphi^\dagger$ as follows:

$$\begin{aligned} \gamma^\dagger &\stackrel{\text{def}}{=} \gamma & (\mathsf{X}\varphi)^\dagger &\stackrel{\text{def}}{=} \mathsf{X}(\alpha \mathsf{U}(\neg\alpha \wedge \varphi^\dagger)) \\ (\neg\varphi)^\dagger &\stackrel{\text{def}}{=} \neg(\varphi^\dagger) & (\varphi_1 \mathsf{U} \varphi_2)^\dagger &\stackrel{\text{def}}{=} (\alpha \vee \varphi_1^\dagger \mathsf{U}(\neg\alpha \wedge \varphi_2^\dagger)) \\ (\varphi_1 \wedge \varphi_2)^\dagger &\stackrel{\text{def}}{=} \varphi_1^\dagger \wedge \varphi_2^\dagger \end{aligned}$$

Clearly, $\varphi^\dagger$ can be constructed by a log-space transducer. The following lemma establishes the correspondence between $\mathcal{A}, \varphi$ and $\mathcal{A}', \varphi^\dagger$.

Lemma 5.2.1 *Let $\mathcal{A}$, φ and (q, n) be an instance of an LTL model checking problem, and let $\mathcal{A}'$ and $\varphi^\dagger$ be defined as above. Then $(T(\mathcal{A}), (q, n)) \models \varphi$ if, and only if, $(T(\mathcal{A}'), (q, n)) \models \varphi^\dagger$.*

Proof. We sketch a proof by structural induction on φ and only consider the interesting cases $\varphi = \mathsf{X}\varphi'$ and $\varphi = \varphi_1 \mathsf{U} \varphi_2$. For $\varphi = \mathsf{X}\varphi'$, suppose $(T(\mathcal{A}), (q, n)) \models \varphi$, by the semantic definition there exists an infinite path ϱ with a trace τ such that $\tau \models \varphi$, *i.e.*, $\tau^1 \models \varphi'$. Let $\varrho(1) = (q', n')$. By the induction hypothesis, $(T(\mathcal{A}'), (q', n')) \models (\varphi')^\dagger$. The construction of $\mathcal{A}'$ ensures that $(T(\mathcal{A}'), (q', n')) \models \neg\alpha$, hence $(T(\mathcal{A}'), (q', n')) \models \neg\alpha \wedge (\varphi')^\dagger$. Moreover, by the construction of $\mathcal{A}'$ there is a finite (q, n)-(q', n') path ϱ' such that $\lambda(\varrho'(i)) = \alpha$ for all $i \in [2, |\varrho'|]$. Consequently, $(T(\mathcal{A}'), (q, n)) \models \mathsf{X}(\alpha \mathsf{U}(\neg\alpha \wedge (\varphi')^\dagger))$, *i.e.*, $(T(\mathcal{A}'), (q, n)) \models \varphi^\dagger$. The converse direction follows analogously.

For the case $\varphi = \varphi_1 \mathsf{U} \varphi_2$, by the semantic definition there exists an infinite path ϱ with a trace τ and $j \in \mathbb{N}$ such that $\tau \models \varphi$, $\tau(i) \models \varphi_1$ for all $i \in [2, j]$ and $\tau(j) \models \varphi_2$. Consequently, $(T(\mathcal{A}), \varrho(i)) \models \varphi_1$ for all $i \in [2, j]$, $(T(\mathcal{A}), \varrho(j)) \models \varphi_2$ and hence by the induction hypothesis $(T(\mathcal{A}'), \varrho(i)) \models \varphi_1^\dagger$ for all $i \in [2, j]$ and $(T(\mathcal{A}'), \varrho(j)) \models \varphi_2^\dagger$. Moreover, the construction of $\mathcal{A}'$ ensures that $(T(\mathcal{A}'), \varrho(i)) \models \neg\alpha$ for all $i \in [2, j]$ and that there are finite $\varrho(i)$-$\varrho(i+1)$ paths ϱ_i for all $i \in [2, j]$ such that $(T(\mathcal{A}'), \varrho_i(k)) \models \alpha$ for all $k \in [1, |\varrho_i| - 1]$. Consequently, $(T(\mathcal{A}'), (q, n)) \models (\alpha \vee \varphi_1^\dagger) \mathsf{U}(\neg\alpha \wedge \varphi_2^\dagger)$, *i.e.*, $(T(\mathcal{A}'), (q, n)) \models (\varphi_1 \mathsf{U} \varphi_2)^\dagger$. The converse direction follows analogously. □

We can now prove the PSPACE upper bound for LTL model checking of one-counter automata. Given an instance $\mathcal{A}$, φ and (q, n) of a model checking problem, we can construct with a PSPACE transducer the unary one-counter automaton $\mathcal{A}'$ corresponding to $\mathcal{A}$, the LTL formula $\varphi^\dagger$, the Büchi automaton $\mathcal{A}_{\neg\varphi^\dagger}$ and the product automaton $\mathcal{A}'' \overset{\text{def}}{=} \mathcal{A}' \times \mathcal{A}_{\neg\varphi^\dagger}$. We have $(T(\mathcal{A}), (q, n)) \models \varphi$ if, and only if, there is a Büchi run in $T(\mathcal{A}'')$ starting in (q, n). Since $|\mathcal{A}''| = \exp(O(|\mathcal{A}|)) \exp(O(|\varphi|)$ and by Proposition 2.5.3 checking for the existence of a Büchi run in a unary one-counter automaton $\mathcal{A}$ is NL-complete, the combined complexity of LTL model checking of one-counter automata is PSPACE-complete.

If we fix φ, we can avoid the construction of a unary one-counter automaton and directly construct the product $\mathcal{A}' \overset{\text{def}}{=} \mathcal{A} \times \mathcal{A}_{\neg\varphi}$ whose size is $|\mathcal{A}'| = O(|\mathcal{A}||\mathcal{A}_{\neg\varphi}|)$. It then follows from Theorem 4.1.1 that model checking LTL on a one-counter automaton for a fixed LTL formula is coNP-complete. Hardness for coNP can easily be derived from the fact that reachability for one-counter automata is NP-hard. The following theorem summarises the results of this section.

Theorem 5.2.1 *LTL model checking of one-counter automata is PSPACE-complete and coNP-complete for a fixed LTL-formula.*

LTL Model Checking of Parametric One-Counter Automata

In this section, we are going to establish the computational complexity of model checking LTL on parametric one-counter automata. We are going to show that the problem is $\mathsf{coNEXPTIME}$-complete in general and coNP-complete for fixed LTL formulae.

The upper bounds follow straightforwardly in the same way as discussed in the previous section. Given an LTL formula φ, a parametric one-counter automaton $\mathcal{A}$ and a configuration (q, n), in order to decide $(T(\mathcal{A}), (q, n)) \not\models \varphi$ we can construct the Büchi automaton $\mathcal{A}_{\neg\varphi}$ and the product automaton $\mathcal{A}' \overset{\text{def}}{=} \mathcal{A} \times \mathcal{A}_{\neg\varphi}$ and then decide whether there exists a valuation ν such that $T(\mathcal{A}^\nu)$ has a Büchi path starting in (q, n). By Theorem 4.2.2, the latter problem is NP-complete, and since $|\mathcal{A}'| = |\mathcal{A}| \exp(O(|\varphi|))$ we get that model checking LTL on parametric one-counter automata is in $\mathsf{coNEXPTIME}$. If φ is fixed, we have $|\mathcal{A}'| = O(|\mathcal{A}|)$, whence model checking is in coNP. Hardness for coNP follows from coNP-hardness of reachability in parametric one-counter automata, which can be checked with a fixed LTL formula.

Theorem 5.2.2 *LTL model checking of parametric one-counter automata is* coNP-*complete for a fixed* LTL*-formula.*

It remains to show that the combined complexity of LTL model checking on parametric one-counter automata is coNEXPTIME-hard. We reduce from the complement of the NEXPTIME-complete problem Succinct 3-SAT [87], which is to decide whether a Boolean formula in 3-CNF given as a circuit is satisfiable.

In order to define Succinct 3-SAT, we now define circuits in an informal way. A more rigorous treatment can for example be found in [100]. A Boolean circuit consists of Boolean gates, AND-, OR- and NOT-gates. The AND- and OR-gates have two inputs and one output, and the NOT-gate has one input and one output. Inputs to and outputs of the gates are Boolean values, *i.e.*, 0s and 1s, and each gate computes a Boolean function, *i.e.*, an AND-gate outputs $b_1 \wedge b_2$ on input $b_1, b_2 \in \{0, 1\}$. A circuit is a collection of Boolean gates in which the outputs of some gates connect to the inputs of other gates such that the resulting graph is acyclic. It is important to mention that the output of a gate can connect to more than one input of another gate. The inputs of the gates of a circuit that are not connected to any output of another gate are called *inputs to the circuit.* Likewise, the outputs of the gates of a circuit that are not connected to the input of a gate are called *outputs of the circuit.* A circuit $\mathcal{C}$ with m inputs and n outputs computes a function $f : \{0,1\}^m \to \{0,1\}^n$. We define the *size* $|\mathcal{C}|$ *of a circuit* $\mathcal{C}$ to be the number of gates it consists of. An example of a Boolean circuit $\mathcal{C}$ of size three is shown in Figure 5.10. It consists of three gates, one AND-, one OR- and one NOT-gate, has three inputs i_1, i_2, i_3 and two outputs o_1, o_2. Hence it computes a function $f_{\mathcal{C}} : \{0,1\}^3 \to \{0,1\}^2$. Clearly, a polynomially space-bounded deterministic Turing machine can evaluate a circuit, *i.e.*, compute the output for a given input. We can now formally define Succinct 3-SAT. An input to Succinct 3-SAT is given by a Boolean circuit $\mathcal{C}$ that encodes a Boolean formula ψ in 3-CNF in $N = \exp(O(|\mathcal{C}|))$ Boolean variables and with $M = \exp(O(|\mathcal{C}|)$ clauses.

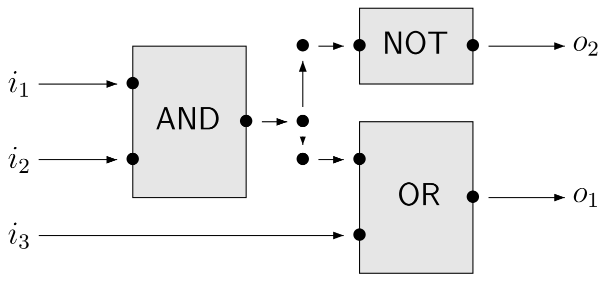

Figure 5.10: An example of a Boolean circuit $\mathcal{C}$ with inputs i_1, i_2, i_3 and outputs o_1, o_2 defining a function $f_{\mathcal{C}} : \{0,1\}^3 \to \{0,1\}^2$. For example, we have $f_{\mathcal{C}}(1,0,1) = (1,1)$ and $f_{\mathcal{C}}(1,1,1) = (0,1)$.

We write ψ as

$$\psi(x_0, \ldots, x_{N-1}) = \bigwedge_{j \in [0,M-1]} (\ell_1^j \vee \ell_2^j \vee \ell_3^j).$$

The circuit $\mathcal{C}$ encodes ψ as follows: $\mathcal{C}$ has $m = (\lg M) + 2$ inputs and $n = (\lg N) + 1$ outputs. On input $c \cdot \ell$, where $c \in \{0,1\}^m$ and $\ell \in \{0,1\}^2$, both read as binary numbers, $f_{\mathcal{C}}$ outputs $x \cdot b$, where $x \in \{0,1\}^n$ and $b \in \{0,1\}$ such that x is the index of the variable of the ℓ-th literal of the clause with index c in ψ, and b indicates whether or not x is negated[4]. Succinct 3-SAT is to decide whether ψ is satisfiable.

Succinct 3-SAT

INPUT: A Boolean circuit $\mathcal{C}$ encoding a Boolean 3-CNF formula ψ.
QUESTION: Is $\psi_{\mathcal{C}}$ satisfiable?

Due to the exponential succinctness provided by Boolean circuits, the complexity of deciding Succinct 3-SAT increases by one exponent as compared to classical 3-SAT.

Proposition 5.2.2 ([87]) Succinct 3-SAT *is* NEXPTIME-*complete.*

In order to establish coNEXPTIME-hardness for the combined complexity of LTL model checking, given an input $\mathcal{C}$ to Succinct 3-SAT, we construct a parametric

[4]For definiteness, ψ can be assumed to be augmented with redundant clauses and literals to handle the cases when $c \geq M$ or $\ell = 4$.

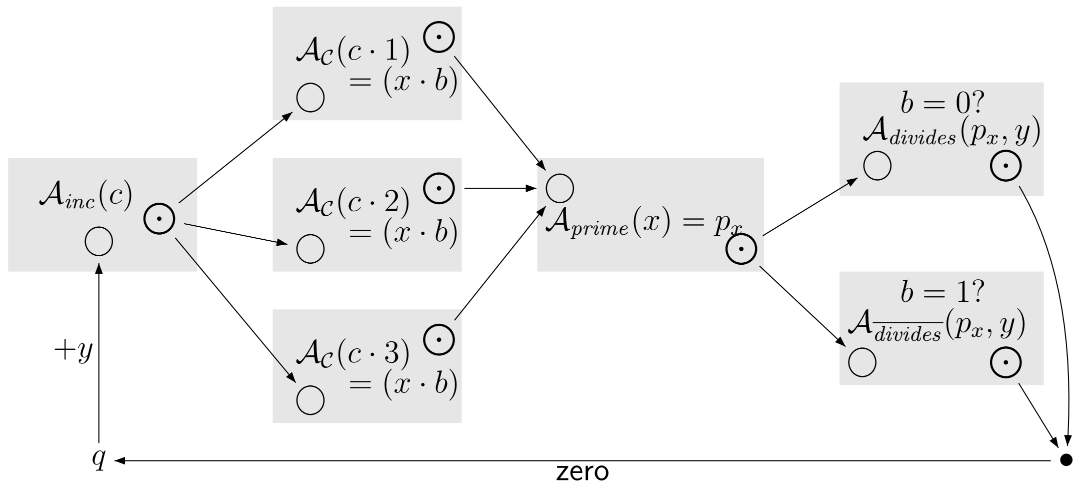
$\mathcal{A}_{inc}(c)$
$\mathcal{A}_{\mathcal{C}}(c \cdot 1) = (x \cdot b)$
$\mathcal{A}_{\mathcal{C}}(c \cdot 2) = (x \cdot b)$
$\mathcal{A}_{\mathcal{C}}(c \cdot 3) = (x \cdot b)$
$\mathcal{A}_{prime}(x) = p_x$
$b = 0?$
$\mathcal{A}_{divides}(p_x, y)$
$b = 1?$
$\mathcal{A}_{\overline{divides}}(p_x, y)$
$+y$
q
zero

Figure 5.11: High-level description of the automaton $\mathcal{A}$ used in the hardness-proof of LTL model checking on parametric one-counter automata.

one-counter automaton $\mathcal{A}$ with one parameter y and an LTL formula φ such that ψ is satisfiable if, and only if, there is a valuation ν such that $(T(\mathcal{A}^\nu), (q, 0)) \models \varphi$ for some distinguished control location q of $\mathcal{A}(y)$.

As a first step, let us provide a suitable encoding of truth assignments of the variables of ψ by natural numbers. The encoding we use has also been employed for establishing lower bounds for EF model checking of unary one-counter automata [68]. Let p_i denote the i-th prime number. Every natural number y defines a truth assignment $\nu : \{x_1, \ldots, x_N\} \to \{0, 1\}$ such that $\nu(x_i) = 1$ if, and only if, $p_i | y$. By the prime number theorem, $p_N = O(N \log N)$ and hence $O(|\mathcal{C}|^2)$ bits are sufficient to represent p_N. We will use the parameter y to guess an assignment, but the above encoding of course requires exponentially many prime numbers to verify that this assignment evaluates ψ to true, and those prime numbers cannot be hardwired into $\mathcal{A}$. Instead, they are going to be computed in $\mathcal{A}$ on-the-fly.

Let us now take a high-level look at $\mathcal{A}$, which is sketched in Figure 5.11. It uses one parameter y and employs several gadgets. The only gadgets manipulating the counter are $\mathcal{A}_{divides}$ and $\mathcal{A}_{\overline{divides}}$. The remaining gadgets are designed in a way such that they communicate via designated propositional variables and not, as in Section 5.1.2, with the help of the counter. First, $\mathcal{A}$ loads the value of the parameter y on the counter.

Recall that the value of y is supposed to encode a truth assignment to $\psi_{\mathcal{C}}$. Next, $\mathcal{A}$ traverses through a gadget $\mathcal{A}_{inc}$, which initially chooses an arbitrary index c identifying a clause of ψ. Every time $\mathcal{A}_{inc}$ is traversed afterwards, it increments c modulo M and hereby moves on to the next clause. Now $\mathcal{A}$ branches non-deterministically into a gadget $\mathcal{A}_{\mathcal{C}}$ in order to compute $x \cdot b$ from $\mathcal{C}$ on input $c \cdot 1, c \cdot 2$, respectively $c \cdot 3$, *i.e.*, in order to compute the index of the variable of the first, second or third literal of the clause with index c. The computed index x is then used as input to a gadget $\mathcal{A}_{prime}$, which computes p_c. Then if $b = 0$, it is checked in $\mathcal{A}_{\overline{divides}}$ that p_i does not divide the value of y, and likewise in $\mathcal{A}_{divides}$ that p_i divides the value of y if $b = 1$. These checks require the counter to be modified. After the checks have been finished, $\mathcal{A}$ restores the value y on the counter and the process continues with clause $c + 1$ mod M. Clearly, if ψ is satisfiable then there exists a valuation ν of y such that there is an infinite path in $T(\mathcal{A}^{\nu})$ that traverses the control location q infinitely often, since every time we traverse $\mathcal{A}_{inc}$ we can always determine which $\mathcal{A}_{\mathcal{C}}$ gadget to choose next so that we do not "get stuck" at the divisibility respectively non-divisibility tests.

It remains to show how the gadgets and the communication between them can be realised. Our first observation is that the computations of $\mathcal{A}_{inc}$, $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{prime}$ can be realised by space bounded deterministic Turing machines using no more than a number of tape cells polynomial in $|\mathcal{C}|$. Indeed, it is easily seen that incrementing modulo M, evaluating $\mathcal{C}$ and computing the i-th prime number p_i can be performed by such a deterministic Turing machine. Thus, we now show how given a generic space-bounded deterministic Turing machine $\mathcal{M}$, we can construct in polynomial time a one-counter automaton $\mathcal{A}_{\mathcal{M}}$ and some LTL formulae that mimic computations of $\mathcal{M}$ on traces of $\mathcal{A}_{\mathcal{M}}$. Our approach is inspired by the classical proof of PSPACE-hardness of LTL model checking on Kripke structures [101].

Let $\mathcal{M} = (S, \Sigma, \Gamma, s_0, A, R, \Delta)$ be a DTM with a fixed input tape with m tape cells, and n working tape cells, and let $S = \{s_0, \ldots, s_k\}$. We may without loss of generality assume that $\Sigma = \Gamma = \{0, 1\}$. Figure 5.12 shows the one-counter automaton $\mathcal{A}_{\mathcal{M}}$ that we use for the simulation of $\mathcal{M}$. There, besides Greek letters we additionally use italic Latin letters in order to denote atomic propositions of $\mathcal{A}_{\mathcal{M}}$. A simulation of $\mathcal{M}$

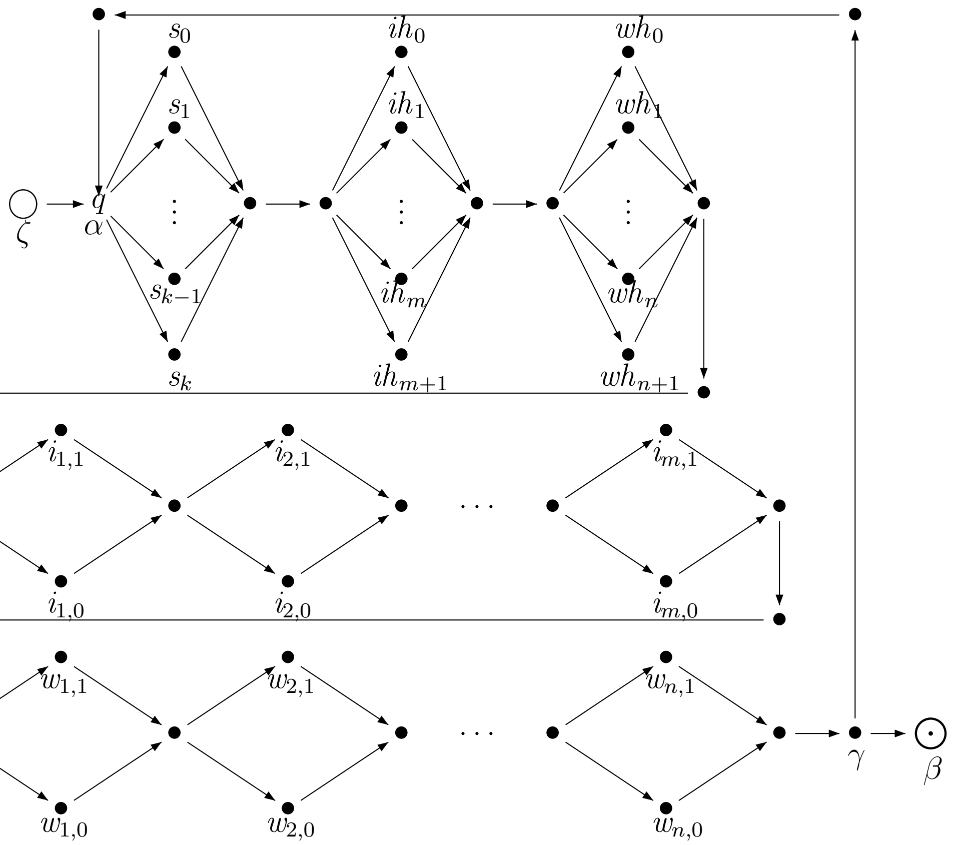

Figure 5.12: One-counter automaton $\mathcal{A}_{\mathcal{M}}$ used for the simulation of a space-bounded deterministic Turing machine.

starts when $\mathcal{A}_{\mathcal{M}}$ is entered at the location labelled with ζ and is finished when the location labelled with β is reached.

The sequence of propositions occurring on a trace of a path starting from and ending in q encodes a configuration of $\mathcal{M}$. In more detail,

- s_i indicates that $\mathcal{M}$ is in state s_i;
- ih_i that the input tape head scans cell i;
- wh_i that the working tape head scans cell i;
- $i_{i,b}$ that the i-th bit of the input tape is set to b; and
- $w_{i,b}$ that the i-th bit of the working tape is set to b, where i is in the respective range and $b \in \{0, 1\}$.

We are now going to introduce some LTL formulae that enforce that a proper sequence of configurations of $\mathcal{M}$ is encoded in traces of $\mathcal{A}_{\mathcal{M}}$. First, we look at LTL formulae that allow for testing properties of the current configuration. It is helpful to think of all of them as being evaluated in q. The formula $\varphi_{state}(i) \overset{\text{def}}{=} \mathsf{X}s_i$ for each $i \in [0, k]$ expresses that the current state of $\mathcal{M}$ is s_i. Additionally, with the formula $\varphi_{inhead}(i) \overset{\text{def}}{=} \mathsf{XXXX}ih_i$ we express that the input head is at position i, where $i \in [0, m+1]$. Similarly, define the formulae $\varphi_{wohead}(i)$, $\varphi_{work}(j, b)$, and $\varphi_{input}(i, b)$ for expressing that the working head is at position i, that the i-th bit of the input tape is b, and that the j-th bit of the working tape is b, respectively, where $i \in [m]$, $j \in [n]$ and $b \in \{0, 1\}$.

The LTL formula below, assumed to be evaluated in the control location labelled with α, ensures that the transition function is correctly encoded into traces of $\mathcal{A}_{\mathcal{M}}$ for states $s \in S \setminus A$ whenever the input respectively working tape head does not scan a start marker $\triangleright$ or end marker $\triangleleft$:

$$\bigwedge_{s_l \in S\setminus A} \bigwedge_{\substack{i\in[m],\\ j\in[n]}} \bigwedge_{b_1,b_2\in\{0,1\}} \varphi_{state}(l) \land \varphi_{inhead}(i) \land \varphi_{input}(i,b_1) \land \varphi_{wohead}(j) \land \varphi_{work}(j,b_2) \rightarrow$$
$$\rightarrow ((\mathsf{X}(\neg\alpha \land \neg\beta)\mathsf{U}(\alpha \land \varphi_{succ}(s_l,i,j,b_1,b_2)) \land$$
$$\land \bigwedge_{\substack{k\neq j,\\ b\in\{0,1\}}} (\varphi_{work}(k,b) \leftrightarrow (\mathsf{X}\neg\alpha\mathsf{U}(\alpha \land \varphi_{work}(k,b)))))).$$

Here, whenever $\Delta(s_l,b_1,b_2) = (s_h,d_1,d_2,b)$, the formula

$$\varphi_{succ}(s_l,i,j,b_1,b_2) \overset{\text{def}}{=} \varphi_{state_h} \land \varphi_{inhead_{i+d_1}} \land \varphi_{wohead_{j+d_2}} \land \varphi_{work_{j,b}}$$

guarantees that the correct bit is "written" to the working tape and that the state, the input head position, and the working tape position of the next configuration seen indeed match the successor configuration. A similar formula can be constructed for the case when one or both of the input or working heads point to a start respectively end marker.

Once we have reached an accepting state $s_i \in A$, we require that $\mathcal{A}_{\mathcal{M}}$ is left, which is expressed by the following formula when evaluated in q:

$$\bigwedge_{s_i\in F} \varphi_{state}(i) \rightarrow (\neg\alpha\mathsf{U}\beta).$$

It is now easily seen that we can construct an LTL-formula $\varphi_{compute}$ that is derived from a conjunction of the formulae from above such that the formula $\mathsf{G}(\alpha \rightarrow \varphi_{compute})$ constraints paths through $\mathcal{A}_{\mathcal{M}}$ in a way such that their traces yield the encoding of a valid computation of $\mathcal{M}$.

Let us now turn towards ensuring that once we enter $\mathcal{A}_{\mathcal{M}}$, we initially traverse it in a way such that the trace corresponds to an initial configuration of $\mathcal{M}$. The formula

$$\mathsf{G}(\zeta \rightarrow \mathsf{X}(\varphi_{state}(0) \land \varphi_{inhead}(1) \land \varphi_{wohead}(1) \land \bigwedge_{j\in[n]} \varphi_{work}(j,0)))$$

ensures that the heads of the input and working tape point to the first tape cell, that the working tape is filled with 0s and that we are in the initial state. If the input tape

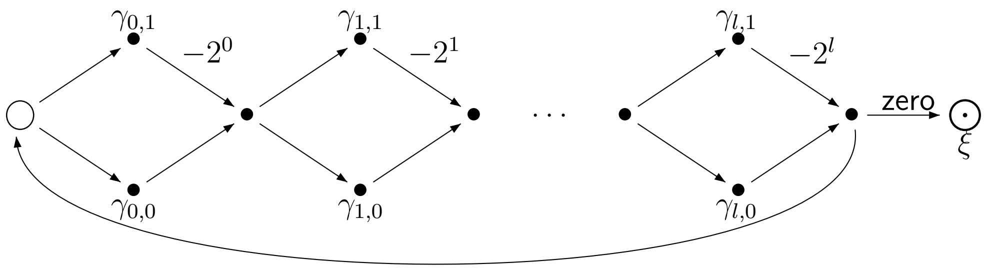

Figure 5.13: The one-counter automaton $\mathcal{A}_{divides}$ for testing the counter for divisibility with some natural number of bit length $l + 1$.

can be initialized with an arbitrary content, we are done. Otherwise, suppose that we want to transfer the first j bits of the output of a space-bounded deterministic Turing machine $\mathcal{M}'$ from its corresponding one-counter automaton $\mathcal{A}_{\mathcal{M}'}$ to the input of $\mathcal{A}_{\mathcal{M}}$. For $b \in \{0, 1\}$, let $\overline{b} \overset{\text{def}}{=} 0$ if $b = 1$ and $\overline{b} \overset{\text{def}}{=} 1$ otherwise, and suppose that all atomic propositions are primed in $\mathcal{A}_{\mathcal{M}'}$. The formula

$$\bigwedge_{\substack{k\in[j]\\ b\in\{0,1\}}} \mathsf{G}((w'_{k,b} \wedge (\neg\alpha' \mathsf{U} \beta')) \rightarrow (\neg i_{k,\bar{b}} \mathsf{U} \gamma)$$

guarantees that we traverse through the first j bits of the component of $\mathcal{A}_{\mathcal{M}}$ representing the input tape of $\mathcal{M}$ in the same way as we traverse the first j bits of the working tape component of $\mathcal{M}'$ in $\mathcal{A}_{\mathcal{M}'}$ when a computation has finished. In summary, we have thus shown how the gadgets $\mathcal{A}_{inc}, \mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{prime}$ from Figure 5.11 and the communication between them can be realised.

The only major question left open is how we can perform a divisibility respectively non-divisibility test of the counter value with a prime number computed in $\mathcal{A}_{prime}$. To this end, let us consider the one-counter automaton $\mathcal{A}_{divides}$ shown in Figure 5.13. One cycle through $\mathcal{A}_{divides}$ subtracts some natural number of bit length $l + 1$ from the counter. In order to properly test for divisibility, we need to make sure that we remain on the same path in every cycle. In the CTL setting, this problem was resolved by branching into the additional one-counter automaton $\mathcal{A}_{bit}$. In contrast, in LTL we cannot branch, but use the propositions $\gamma_{j,b_j}, j \in [0, l], b \in \{0, 1\}$ in order to stay on a precisely fixed path in every cycle. Assuming that the number p for which we want to test for divisibility with the current counter value is encoded as a sequence of propositions w_{j,b_j} of a one-counter automaton $\mathcal{A}_{\mathcal{M}}$, the subsequent formula enforces that we always subtract p in cycles of $\mathcal{A}_{divides}$:

$$\mathsf{G}(\bigwedge_{\substack{j\in[0,l]\\ b_j\in\{0,1\}}}((w_{j,b_j}\wedge(\neg\alpha\mathsf{U}\beta))\rightarrow(\neg\gamma_{j,\overline{b_j}}\mathsf{U}\xi))).$$

It is straightforward to derive a similar one-counter automaton $\mathcal{A}_{\overline{divides}}$ and an appropriate LTL formula for testing non-divisibility of the counter value with a previously computed prime number. Finally, we can also adopt these techniques in order to correctly handle the branching on b performed in Figure 5.11.

In summary, by taking the disjoint union of all the gadgets from Figure 5.11, their appendent LTL formulae that we described in this section, wiring the gadgets correctly and taking the conjunction of the relevant LTL formulae, given an input $\mathcal{C}$ to SUCCINCT 3-SAT, we can construct a parametric one-counter automaton $\mathcal{A}$ with one parameter y and an LTL formula φ such that there is an assignment ν assigning a natural number to y such that $(T(\mathcal{A}^{\nu}),(q,0))\not\models\varphi$ if, and only if, ψ encoded by $\mathcal{C}$ is satisfiable.

Theorem 5.2.3 *The combined complexity of* LTL *model checking on parametric one-counter automata is* coNEXPTIME*-complete.*

5.3 Discussion

This chapter established complexity results for model-checking problems on transition systems generated by one-counter automata and parametric one-counter automata. We considered two classes of specification logics, branching-time and linear time logics.

Section 5.1 dealt with the branching-time logic CTL and its syntactic fragment EF. We showed that the expressive power of those logics renders the model checking problem on parametric one-counter automata undecidable. Decidability of the problem on one-counter automata was already known, the contribution of this chapter is that we have developed tight bounds for the complexity of model checking this class of counter automata. Although EF model checking of one-counter automata with updates encoded in unary is P^{NP}-complete, the succinct encoding of numbers in our model does not increase the complexity by one exponent and model checking EF on one-counter automata is PSPACE-complete. The crucial insight in establishing the upper bound was to show a periodicity property for reachability relations and EF formulae. In contrast, CTL turned out to be expressive enough to fully exploit the exponential succinctness of one-counter automata: while CTL model checking of one-counter automata with updates encoded in unary is PSPACE-complete, it is EXPSPACE-complete already for a fixed formula when updates are encoded in binary. Proving the lower bound was far from being straightforward and shown by using the fact that EXPSPACE is exponentially L-serialisable. In Section 5.2 we then considered model checking LTL on one-counter automata and parametric one-counter automata. In contrast to the branching-time logics, both problems were shown to be decidable. Moreover, we showed that model checking LTL on transition systems generated by one-counter automata is from a complexity point-of-view not harder than standard LTL model checking and PSPACE-complete. Similar to the CTL case, LTL model checking of parametric one-counter automata turned out to be computationally harder, but remained decidable and coNEXPTIME-complete.

We have recently considered in [49] an even more restricted fragment of CTL which only allows for the EX modality. It is not difficult to adjust the proof of Proposition 5.1.2 in order to show that model checking this logic on one-counter automata is PSPACE-hard. For parametric one-counter automata however, model checking becomes decidable and can be shown to be PSPACE-complete.

With regards to future work, it would be interesting to investigate synthesis problems for LTL. Instead of asking whether an LTL formula holds for all one-counter automata obtained from all possible valuations, we could instead ask whether there exists a valuation such that an LTL formula holds. This problem is closely related to the Büchi synthesis problem discussed at the end of the previous chapter. For CTL respectively EF, this problem is undecidable by Theorem 5.1.2 and 5.1.4.

Part II:
On the Verification of Programs with Pointers and Linked Lists

Chapter 6

Tractable Reasoning in a Fragment of Separation Logic

This chapter is about the complexity of reasoning in a fragment of separation logic (SL). Separation logic [66, 92] is an extension of Hoare logic to reason about pointer manipulating programs. It extends the syntax of assertions with predicates describing shapes of memory; aliasing and disjointness can be concisely expressed within these shapes. This extended assertion language allows elegant and concise hand written proofs of programs that manipulate dynamically allocated data structures. However, generating such proofs in an automated fashion is constrained by the undecidability of almost all reasoning tasks in separation logic [92]. For that reason, there has been a lot of research on finding decidable fragments of this logic, see *e.g.* [11, 23].

In this chapter, we study the separation logic fragment introduced by Berdine, Calcagno and O'Hearn in [11]. This chapter is independent from the previous chapters of this thesis. Although in the literature decidability results for separation logic have been obtained via reductions to decision problems for counter automata, as discussed in the introduction of this thesis, we do not follow this approach in this chapter. The fragment of separation logic that we consider and has been presented in [11] allows for reasoning about structural integrity properties of programs with pointers and linked lists. Traditionally, separation logic formulae are interpreted in *memory models* consisting of a *stack* and a *heap*. The stack is a mapping from a finite set of stack variables to cells of the heap. The heap is partitioned into a finite set of *allocated heap cells* and an infinite set of non-allocated heap cells. An allocated heap cell maps, or in different words *points to*, a possibly non-allocated heap cell. The separation logic assertion language we consider allows for specifying shapes of the heap. An assertion consists of a *pure* and a *spatial part*. The pure part consists of a conjunction of equalities and disequalities over the stack variables. For example, the pure assertion $x = y \wedge x \neq z$ holds in all memory models in which the stack variables x and y map to the same heap cell, and in which the heap cell that x (and thus y) maps to is different from the cell that z maps to. An atomic spatial assertion is either $x \mapsto y$ or $\ell s(x, y)$, where x and y are stack variables. The semantics is that $x \mapsto y$ holds in a memory model in which the heap cell that x maps to is allocated and points to the heap cell of y, which is not required to be allocated. The assertion $\ell s(x, y)$ holds in a memory model if either the heap cell that x is mapped to is equivalent to the heap cell of y and no heap cell is allocated, *i.e.*, there is an empty list on the heap, or there is a chain of heap cells $c_1, \dots, c_{n+1}$ such that $c_1, \dots, c_n$ are allocated and c_i points to c_{i+1} for all $i \in [n]$. Atomic spatial assertions can be combined with the *star-operator*. Given atomic spatial formulae σ_1, σ_2, we have that $\sigma_1 * \sigma_2$ holds in a memory model if its set of allocated heap cells can be separated into two disjoint parts such that in one part σ_1 holds and σ_2 holds in the other part. Recall that, for example, $x \mapsto y$ does not require y to be allocated in a memory model. Therefore, both σ_1 and σ_2 can mention the same stack variable without resulting in an inconsistent spatial assertion. So, for example, $x \mapsto y * \ell s(y, z)$ has a memory model, even though y is mentioned on both sides of the star-operator. An assertion in the separation logic fragment that we consider now is a tuple consisting of a pure and a spatial formula. For example,

$$\alpha = (y \neq z; x \mapsto y * \ell s(y, z))$$

describes memory models in which y and z map to different heap cells, and in which the heap can be separated into disjoint heaps, on in which the heap cell of x is allocated and maps to the heap cell of y, and another heap in which there is a list,*i.e.* a possibly empty chain of allocated heap cells forming a linked list, from y to z. Since

y and z are required to be disjoint, this list is non-empty and thus y is required to be allocated in the heap in which $\ell s(y, z)$ holds.

The decision problem that we consider in this chapter is *entailment*, which is to decide given assertions α_1, α_2 whether α_2 holds in every memory model in which α_1 holds. We write $\alpha_1 \models \alpha_2$ if α_1 entails α_2 and show that entailment can be decided in polynomial time. For example, we have $x \mapsto y \models \ell s(x, y)$, but $\ell s(x, y) \not\models x \mapsto y$ since the list from x to y could possibly be empty or of length greater than two. Thus, a memory model disproving entailment can be obtained by considering the canonical memory model obtained from replacing the $\ell s(x, y)$ assertion with two new assertions $x \mapsto z * z \mapsto y$, where z is a fresh stack variable. A formalisation of this way of disproving entailment was developed in [11] by showing that list assertions need to be expanded to length at most two in order to disprove that an entailment holds. This immediately yields a coNP algorithm for entailment which, in order to disprove an entailment, non-deterministically guesses how much any list assertion on the left-hand side of an entailment needs to be expended.

In order to show that entailment is computable in polynomial time, we need to take a fundamentally different approach to [11]. The first difference is that we represent memory models and assertions as a special class of directed coloured graphs, which we call *SL graphs*. In order to represent memory models, in an SL graph, heap cells are nodes which are coloured red if they are allocated and coloured black if they are not allocated. Each node is labelled with a finite set of stack variables that point to this heap cell this node represents. Special edges between nodes allow for indicating that heap cells are disjoint. Arrows between cells indicate that the heap cell at the source of the arrow points to the heap cell at the target of the arrow. Figure 6.1(a) shows an example of how we graphically represent memory models. There, nodes coloured red are circles, and thus the represented memory model consists of three allocated heap cells, to which the stack variables x, y and u point to. The heap cell with stack variable x points to the heap cell with stack variable y, *etc.* Dashed lines explicitly assert that the heap cells are not equivalent, *e.g.*, a dashed line between the heap cell labelled with x and the heap cell labelled with y asserts that those heaps

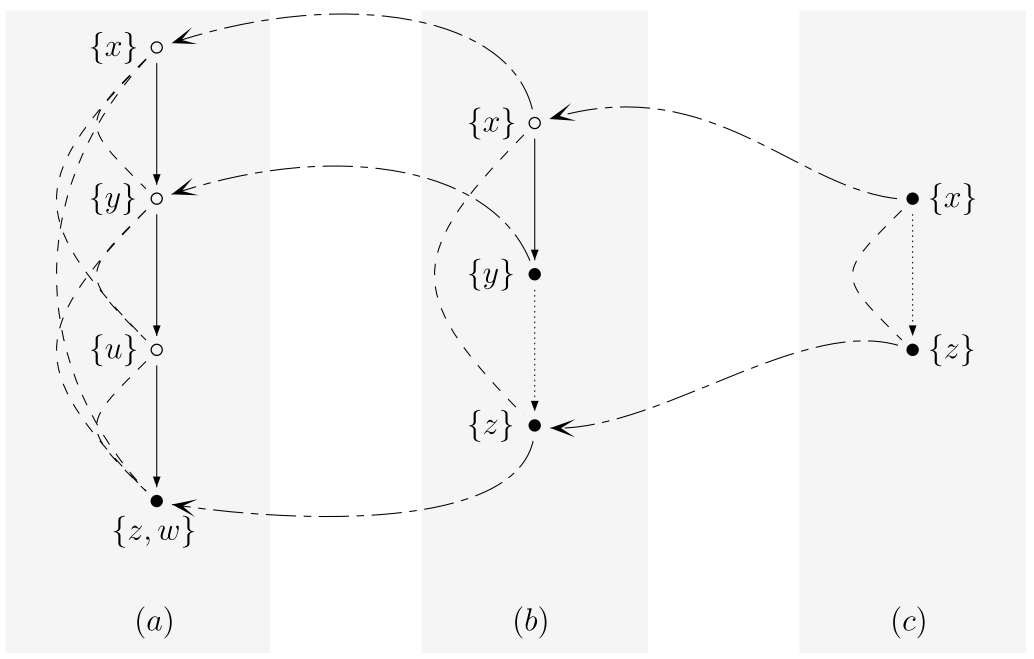

Figure 6.1: Examples of the way memory models and assertions are graphically represented in this chapter and homomorphic embeddings.

cells are not equivalent. Finally, there is one non-allocated heap cell to which the stack variables z and w point, *i.e.*, z and y are equivalent in the represented memory model. Some structural restrictions have to be made in order to reflect particularities of the semantics of separation logic, but this basically is the main idea how memory models are represented. Separation logic assertions are also represented as graphs, but allow for additional list edges between nodes, depicted as dotted arrows. Figure 6.1(b) and (c) show the graphs corresponding to $\alpha_1 = (x \neq z; x \mapsto y * \ell s(y, z))$ respectively $\alpha_2 = (x \neq z; \ell s(x, z))$. The $\ell s(y, z)$ assertion is represented as a dotted list in the figure. The graphs are constructed in a way such that each SL assertion has a corresponding SL graph and *vice versa*. So in particular, a memory model also corresponds to an SL formula. The advantage of representing assertions as graphs and memory models as a subclass of general SL graphs is that entailment can be decided by checking for the existence of a homomorphic embedding, which is a mapping between the node of SL graphs that preserves structural properties. For example, the memory model in (a) is a model of the assertion α_1, and an embedding of the SL graph corresponding to α_1 is shown in Figure 6.1 by the arrows from (b) to (a). The homomorphism is fully determined by the stack variables, and once it has been fixed, it can be checked that all assertions are fulfilled. Indeed, $x \neq z$ holds in the memory model, x points to y as well and there is a list from y to z. Likewise, there is a homomorphism from the graph of α_2 to the graph of α_2, and we are going to show in this chapter that this implies that $\alpha_1 \models \alpha_2$. The difficult part is that, in general, computing homomorphisms between graphs is an NP-complete problem. However, we are going to show that we can bring SL graphs into a particular normal form, where deciding the existence of a homomorphism can be performed in polynomial time.

The fragment of separation logic that we consider is the basis for tools such as Smallfoot [12], which however employs the coNP algorithm discussed above in order to decide entailments. Despite the worst-case exponential time complexity, the tool demonstrated that separation logic could be used to automatically verify memory safety of linked list and tree manipulating programs. Based on the success of Smallfoot, this approach has been extended to allow automatic inference of specifications of systems code [10, 22], to reason about object-oriented programs [43, 67], and even to reason about non-blocking concurrent programs [12]. But fundamentally all these tools are based on the same style of syntactic proof theory presented in [11].

This chapter is structured as follows: in Section 6.1 we formally introduce our fragment of separation logic, graphs and the decision problems that we consider. Section 6.2.1 then shows how we can compute in polynomial time from a given assertion a graph in normal form that represents the same set of models of the formula. We then show in Section 6.2 that a homomorphism between graphs in normal form witnesses an entailment, and that such a homomorphism can be computed in polynomial time. Section 6.3 deals with syntactic extensions that make entailment coNP-hard. We close this chapter with a discussion in Section 6.4, where we in particular focus on the differences between the semantic model used in thesis and in [11].

6.1 Separation Logic and SL-Graphs

Let *Vars* and V be countably infinite sets of *variables* and *nodes*. We assume some fixed total order $<$ on *Vars* and for any finite set $S \subseteq \mathit{Vars}$, we denote by $\min(S)$ the unique $x \in S$ such that $x \leq y$ for all $y \in S$.

The syntax of our assertion language is given by the following grammar, where x ranges over *Vars*:

$$
\begin{array}{rll}
\mathit{expr} & ::= x & (\mathit{expressions}) \\
\phi & ::= \mathit{expr} = \mathit{expr} \mid \mathit{expr} \neq \mathit{expr} \mid \phi \wedge \phi & (\mathit{pure\ formulae}) \\
\sigma & ::= \mathit{expr} \mapsto \mathit{expr} \mid \ell s(\mathit{expr}, \mathit{expr}) \mid \sigma * \sigma & (\mathit{spatial\ forumlae}) \\
\alpha & ::= (\phi; \sigma) & (\mathit{assertions})
\end{array}
$$

Subsequently, we call formulae of our assertion language *SL-formulae*. An example of an SL-formula is $\alpha = (x \neq y; \ell s(x,y) * y \mapsto z)$. It describes memory models in which the value of the stack variable x is not equal to the value of the stack variable y, and in which the heap can be separated into two disjoint segments such that in one segment there is a linked list from the heap cell whose address is the value of x to the heap cell whose address is the value of y, and where in the other segment the latter heap cell points to the heap cell whose address is z. We denote by $|\phi|$ the *size* of a pure formula and by $|\sigma|$ the size of a spatial formula, which is in both cases the number of symbols used to write down the formula. Given an assertion $\alpha = (\phi; \sigma)$, the size of α is $|\alpha| \overset{\text{def}}{=} |\phi| + |\sigma|$. By ϵ, we subsequently denote the empty spatial assertion of size zero.

The semantics of SL-formulae is given in terms of SL-graphs, which we define to be a special class of directed graphs. Later, we are also going to use SL-graphs in order to represent SL-formulae.

Definition 18 An *SL-graph* G is either $\bot$ or $(V_b, V_r, E_l, E_p, E_d, \ell)$ such that

- $V_b, V_r \subseteq_{\text{fin}} V$, $V_b \cap V_r = \emptyset$, $V_{b,r} \overset{\text{def}}{=} V_b \cup V_r$;
- $E_l \subseteq V_{b,r} \times V_{b,r}$;

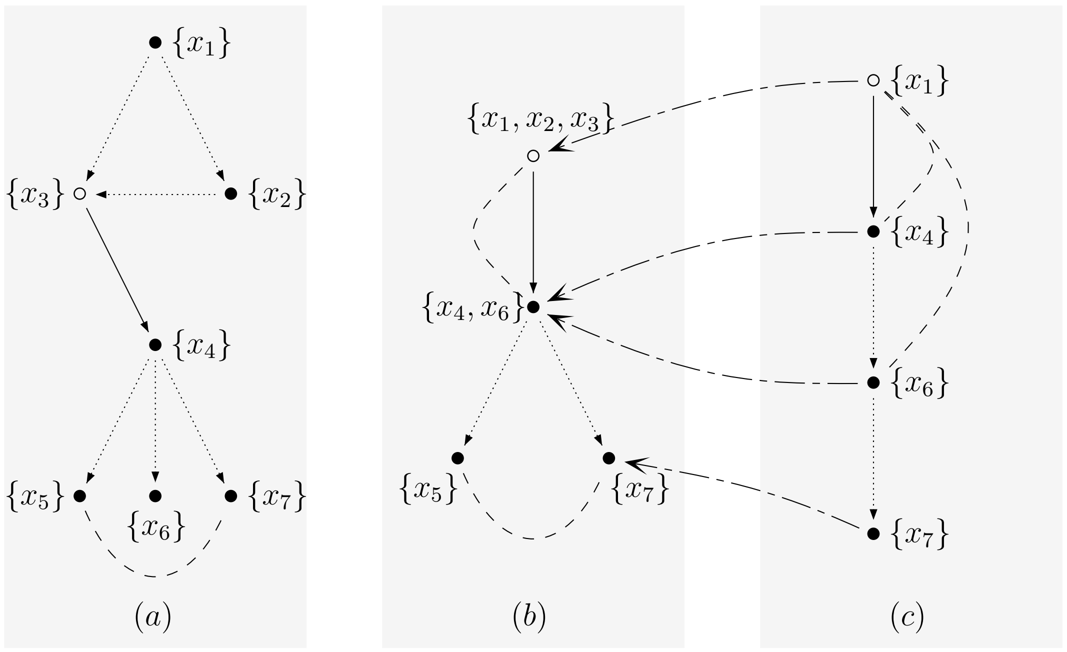

Figure 6.2: Three SL-graphs, where bullets represent black nodes, circles red nodes and where l-edges are dotted arrows, p-edges solid arrows and d-edges dashed lines. Nodes are labelled with the variables next to them. The graphs (b) and (c) are in normal form, where (b) is obtained by reducing (a). The arrows from (c) to (b) depict a homomorphism.

- $E_p \subseteq V_r \times V_{b,r}$ and for every $v \in V_r$, $E_p(v) \neq \emptyset$;
- $E_d \subseteq \{\{v, w\} : v, w \in V_{b,r}, v \neq w\}$; and
- $\ell : \mathit{Vars} \rightharpoonup_{\text{fin}} V_{b,r}$

An *SL-interpretation* is an SL-graph where $E_l = \emptyset$, E_p is functional and $E_d = \{\{v, w\} : v, w \in V_{b,r}, v \neq w\}$. ◊

An SL-graph $\perp$ indicates an inconsistent SL-graph. The set $V_{b,r}$ of *nodes* of an SL-graph partitions into sets V_b and V_r, where we refer to nodes in V_b as *black nodes* and to those in V_r as *red nodes*. We call E_p the set of *pointer edges (p-edges)*, E_l the set of *list edges (l-edges)*, E_d is the set of *disequality edges (d-edges)* and ℓ the *variable*

$$
\begin{aligned}
\mathcal{I} \models x = y &\iff \ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)\\
\mathcal{I} \models x \neq y &\iff \ell^{\mathcal{I}}(x) \neq \ell^{\mathcal{I}}(y)\\
\mathcal{I} \models \phi_1 \wedge \phi_2 &\iff \mathcal{I} \models \phi_1 \text{ and } \mathcal{I} \models \phi_2\\
\mathcal{I} \models x \mapsto y &\iff \exists v, w \in V^{\mathcal{I}}_{b,r}.V^{\mathcal{I}}_r = \{v\}, E^{\mathcal{I}}_p = \{(v,w)\}, \ell^{\mathcal{I}}(x) = v, \ell^{\mathcal{I}}(y) = w\\
\mathcal{I} \models \ell s(x,y) &\iff \exists n \in \mathbb{N}.\mathcal{I} \models \ell s^n(x,y)\\
\mathcal{I} \models \ell s^0(x,y) &\iff \ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y) \text{ and } V^{\mathcal{I}}_r = \emptyset\\
\mathcal{I} \models \ell s^{n+1}(x,y) &\iff \exists z \notin dom(\ell^{\mathcal{I}}), v \in V.\mathcal{I}[\ell[z \mapsto v/\ell]] \models x \mapsto z * \ell s^n(z,y)\\
\mathcal{I} \models \sigma_1 * \sigma_2 &\iff \exists \mathcal{I}_1, \mathcal{I}_2.\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2, \mathcal{I}_1 \models \sigma_1, \mathcal{I}_2 \models \sigma_2\\
\mathcal{I} \models (\phi;\sigma) &\iff \mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2, \mathcal{I}_1 \models \phi \text{ and } \mathcal{I}_1 \models \sigma, \text{ where } \mathcal{I} \models \epsilon \text{ for all } \mathcal{I}
\end{aligned}
$$

Table 6.1: Semantics of the assertion language, where $\mathcal{I}$ is an SL-interpretation.

labelling function. For convenience, $E_{p,l}$ denotes the set $E_p \cup E_l$. Given a node $v \in V$, we set $\text{vars}(v) \overset{\text{def}}{=} \{x \in \mathit{Vars} : \ell(x) = v\}$ and $\text{var}(v) \overset{\text{def}}{=} \min(\text{vars}(v))$. We sometimes wish to alter one component of a graph and, *e.g.*, write $G[E'_p/E_p]$ to denote the graph $G' = (V_b, V_r, E'_p, E_l, E_d, \ell)$.

Figure 6.2 shows three examples of SL-graphs. Subsequently, we identify nodes of an SL-graph with any of the variables they are labelled with. Graph (a) has an l-edge from the black node x_1 to the red node x_3, depicted by a dotted arrow. The latter node has a p-edge to the black node x_4, depicted by a solid arrow. Moreover, there is a d-edge between x_5 and x_7, depicted by a dashed line.

In the remainder of this chapter, we denote an SL-interpretation by $\mathcal{I}$ and usually denote the components of an interpretation with superscript $\mathcal{I}$, *e.g.*, we write $V^{\mathcal{I}}_b$ in order to denote the black nodes of an interpretation $\mathcal{I}$. Given SL-interpretations $\mathcal{I}, \mathcal{I}', \mathcal{I}''$, we define the *star operator* as $\mathcal{I} = \mathcal{I}' * \mathcal{I}''$ if, and only if,

- $V^{\mathcal{I}}_r = V^{\mathcal{I}'}_r \uplus V^{\mathcal{I}''}_r$;
- $V^{\mathcal{I}'}_b = V^{\mathcal{I}}_b \cup V^{\mathcal{I}''}_r$;

- $V_b^{\mathcal{I}''} = V_b^{\mathcal{I}} \cup V_r^{\mathcal{I}'}$;
- $E_p^{\mathcal{I}} = E_p^{\mathcal{I}'} \uplus E_p^{\mathcal{I}''}$; and
- $\ell^{\mathcal{I}} = \ell^{\mathcal{I}'} = \ell^{\mathcal{I}''}$.

The semantics of our assertion language is presented in Table 6.1. We call $\mathcal{I}$ a *model* of α if $\mathcal{I} \models \alpha$. The decision problems of interest to us are *satisfiability* and *entailment*.

<u>SL Satisfiability</u>

INPUT: An assertion α.
QUESTION: Does there exist an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \alpha$?

<u>SL Entailment</u>

INPUT: Assertions α, α'.
QUESTION: Does $\mathcal{I} \models \alpha$ imply $\mathcal{I} \models \alpha'$ for all interpretations $\mathcal{I}$?

Given an assertion α, we say α *is satisfiable* if there exists a model $\mathcal{I}$ such that $\mathcal{I} \models \alpha$. Given two assertions α_1 and α_2, we say α_1 *entails* α_2 if for any SL-interpretation $\mathcal{I}$, whenever $\mathcal{I} \models \alpha_1$ then $\mathcal{I} \models \alpha_2$. We write $\alpha_1 \models \alpha_2$ if α_1 entails α_2, and $\alpha_1 \equiv \alpha_2$ if $\alpha_1 \models \alpha_2$ and $\alpha_2 \models \alpha_1$.

Given an SL-graph G, we are now going to define its *corresponding assertion* $\alpha(G)$. If $G = \bot$ then $\alpha(G) \stackrel{\text{def}}{=} (x \neq x; \epsilon)$, *i.e.*, an unsatisfiable SL-formula. Otherwise, the assertion $\alpha(G)$ corresponding to G is defined as follows, where we use an indexed star operator:

$$\begin{aligned}
\phi(G) &\stackrel{\text{def}}{=} \bigwedge_{\substack{v \in V_{b,r} \\ x,y \in \text{vars}(v)}} x = y \wedge \bigwedge_{\{v,w\} \in E_d} \text{var}(v) \neq \text{var}(w), \\
\sigma(G) &\stackrel{\text{def}}{=} \left(*_{(v,w) \in E_p} \text{var}(v) \mapsto \text{var}(w) \right) * \left(*_{(v,w) \in E_l} \ell s(\text{var}(v), \text{var}(w)) \right), \\
\alpha(G) &\stackrel{\text{def}}{=} (\phi(G), \sigma(G)).
\end{aligned}$$

We define the *size* of an SL-graph G as $|G| \stackrel{\text{def}}{=} |\alpha(G)|$. An example of the above definition is given in Figure 6.2, where graph (b) corresponds to the assertion $\alpha =$

$(\phi; \sigma)$, where

$$\begin{aligned}
\phi &\overset{\text{def}}{=} x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_6 \wedge x_1 \neq x_4 \wedge x_5 \neq x_7, \\
\sigma &\overset{\text{def}}{=} x_1 \mapsto x_4 * \ell s(x_4, x_5) * \ell s(x_4, x_7)),
\end{aligned}$$

and where we have omitted superfluous equalities.

We now give some technical definitions about paths in SL-graphs. Given a relation $E \subseteq V \times V$, a *v-w path in E of length n* is a sequence of nodes $\pi : v_1 \cdots v_{n+1}$ such that $v_1 = v$, $v_{n+1} = w$ and $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq n$. We write $|\pi|$ to denote the length of π. The *edges traversed by* π is defined as $\text{edges}(\pi) \overset{\text{def}}{=} \{(v_i, v_{i+1}) : 1 \leq i \leq n\}$. Two paths π_1, π_2 are *distinct* if $\text{edges}(\pi_1) \cap \text{edges}(\pi_2) = \emptyset$. If $v \neq w$, we call a v-w path *loop-free* if $v_i \neq v_j$ for all $1 \leq i \neq j \leq n+1$. We write $v \rightsquigarrow_p w$, $v \rightsquigarrow_l w$ and $v \rightsquigarrow_{p,l} w$ if there exists a v-w path in E_p, E_l respectively $E_{p,l}$. Moreover, we write $v \rightarrow_p w$, $v \rightarrow_l w$ and $v \rightarrow_{p,l} w$ if $(v, w) \in E_p$, $(v, w) \in E_l$ respectively $(v, w) \in E_{p,l}$. Given a set of edges E, $V(E)$ denotes the set $V(E) \overset{\text{def}}{=} \{v : \exists w.(v, w) \in E \text{ or } (w, v) \in E\}$. As usual, E^* denotes the reflexive and transitive closure of E. For $e = (v, w) \in E$, we define $E^*(e) \overset{\text{def}}{=} \{u : (w, u) \in E^*\} \cup \{v\}$, *i.e.*, $E^*(e)$ is the set of all nodes reachable starting from edge e.

6.2 Deciding Entailment via Homomorphisms between SL-Graphs

The challenging aspect in giving a polynomial time algorithm to decide entailment is that there is some implicit non-determinism introduced by list assertions. As has already been observed in [11], given $\alpha = (y \neq z; \ell s(x, y) * \ell s(x, z))$, for any model $\mathcal{I}$ of α we have $\mathcal{I} \models (x = y; \epsilon)$ or $\mathcal{I} \models (x = z; \epsilon)$. However there are models $\mathcal{I}_1, \mathcal{I}_2$ of α such that $\mathcal{I}_1 \not\models (x = y; \epsilon)$ and $\mathcal{I}_2 \not\models (x = z; \epsilon)$. Non-determinism often makes computing entailment coNP-hard for logics that contain predicates for describing reachability relations on graphs, *e.g.*, in fragments of XPath or description logics [79, 55]. However, in our SL fragment we obtain tractability through the SL-graph normal form we are going to develop in the next section and the fact that variable names only occur at exactly one node in an SL-graph, which fully determines a graph homomorphism if it exists.

The structure of this section is as follows. The first part deals with a particular normal form of SL-graphs. We are going to show that any satisfiable SL-formula has an equivalent SL-formula whose corresponding SL graph is in such a normal form and which can be computed in polynomial time. The subsequent section then shows that entailment can be decided by checking for the existence of a homomorphism between SL-graphs in normal form. The key property is that a homomorphism can be computed in polynomial time, which yields a polynomial-time algorithm for checking entailment between SL formulae.

6.2.1 A Normal form of SL-Graphs

In this section, we are going to show that given an assertion α, we can compute in polynomial time an SL-graph G in a normal form such that $\alpha \equiv \alpha(G)$. This normal form serves three purposes:

- it makes implicit equalities and disequalities from α explicit;
- an SL-graph in normal form has the structural property that if there is a loop-free path between two distinct vertices then there is exactly one such path; and
- any SL-graph $G \neq \bot$ in normal form can be transformed into an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \alpha(G)$, thus showing that satisfiability in our SL fragment is in polynomial time.

Our strategy is as follows: we first show how given an assertion α, we can compute an SL-graph G such that $\alpha \equiv \alpha(G)$. Next, we define the normal form of SL-graphs and show that from any SL-graph $G \neq \bot$ we can compute an SL-graph G' in normal form such that $\alpha(G) \equiv \alpha(G')$.

To begin with, we show how given a pure formula ϕ we can construct a corresponding graph G_ϕ such that $(\phi, \epsilon) \equiv \alpha(G_\phi)$. Let $\{x_1, \ldots, x_m\} \subseteq \mathit{Vars}$ be the set of all variables occurring in ϕ, and let $\{[e_1], \ldots, [e_n]\}$ be the set of all equivalence classes of variables induced by ϕ, *i.e.*, $x, y \in [e_i]$ if, and only if, ϕ implies $x = y$. Let $V_b \overset{\text{def}}{=} \{v_1, \ldots, v_n\} \subseteq V$; $\ell(x) \overset{\text{def}}{=} v_i$ if, and only if, $x \in [e_i]$; and $E_d \overset{\text{def}}{=} \{\{v_i, v_j\} : \exists x, y \in \mathit{Vars}. x \in [e_i], y \in [x_j]$ and $x \neq y$ occurs in $\phi\}$. If there is a singleton set in E_d then set $G_\phi \overset{\text{def}}{=} \bot$, otherwise $G_\phi \overset{\text{def}}{=} (V_b, \emptyset, \emptyset, \emptyset, E_d, \ell)$. The following lemma can now easily be verified.

Lemma 6.2.1 *Let ϕ be a pure formula. There exists a polynomial time computable SL-graph G_ϕ such that $\alpha(G_\phi) \equiv (\phi, \epsilon)$.*

Next, we show how to deal with spatial assertions. When processing spatial assertions and transforming SL-graphs into normal form, we need to manipulate SL-graphs. The two operations we perform on them are *merging nodes* and *removing edges*. These operations can be realised by the algorithms that we subsequently introduce. Let us fix an SL graph $G = (V_b, V_r, E_l, E_p, E_d, \ell)$. Algorithm $\textsc{Merge}(G, v, w)$ takes an SL-graph G as input and merges the node w into node v by adding all labels from w to the labels of v and appropriately updating E_l, E_p and E_d. Moreover, the algorithm makes sure that if either $v \in V_r$ or $w \in V_r$ then $v \in V_r$ in the returned graph. If both $v, w \in V_r$ or $\{v, w\} \in E_d$ then $\textsc{Merge}(G, v, w)$ returns $\bot$. It is obvious that the algorithm runs in polynomial time, and we can characterise $\textsc{Merge}$ as follows.

Lemma 6.2.2 *Let $\alpha(G) = (\phi; \sigma)$, $v, w \in V_{b,r}$, $x = \mathrm{var}(v)$ and $y = \mathrm{var}(w)$. We have $\alpha(\textsc{Merge}(G, v, w)) \equiv (\phi \wedge x = y; \sigma)$.*

In order to remove edges, we define for removing l- and p-edges. Algorithm $\textsc{LRemove}(G, (v, w))$ takes an SL-graph G and an l-edge as input and removes the (v, w) from E_l. We omit the pseudo-code of $\textsc{LRemove}$ for readability, the algorithm can be characterised as follows.

Lemma 6.2.3 *Let $\alpha(G) = (\phi; \sigma * \ell s(x, y))$, $v, w \in V_{b,r}$, $x = \mathrm{var}(v)$ and $y = \mathrm{var}(w)$. We have $\alpha(\textsc{LRemove}(G, (v, w))) \equiv (\phi; \sigma)$.*

Similarly to $\textsc{LRemove}$, $\textsc{PRemove}(G, (v, w))$ removes a p-edge (v, w) from E_p and, if necessary, moves v from V_r to V_b if v has as a result no outgoing p-edge. The algorithm can be characterised as follows.

Algorithm 4 $\textsc{Merge}(G, v, w)$ merging two nodes v and w of G semantically

```
Input: An SL-graph $G$ and nodes $v, w \in V_{b,r}$
  if $v = w$ then
    return $G$
  end if
  if $\{v, w\} \subseteq V_r$ or $\{v, w\} \in E_d$ then
    return $\perp$
  end if
  $V'_b \overset{\text{def}}{=} V_b \setminus \{w\}; V'_r \overset{\text{def}}{=} V_r \setminus \{w\}$
  if $w \in V_r$ then
    $V'_r \overset{\text{def}}{=} V'_r \cup \{v\}; V'_b \overset{\text{def}}{=} V'_b \setminus \{v\}$
  end if
  $E'_p \overset{\text{def}}{=} E_p \setminus (\{(w', w) \in E_p\} \cup \{(w, w') \in E_p\})$
  $E'_p \overset{\text{def}}{=} E'_p \cup (\{(w', v) : (w', w) \in E_p\} \cup \{(v, w') : (w, w') \in E_p\})$
  $E'_l \overset{\text{def}}{=} E_l \setminus (\{(w', w) \in E_l\} \cup \{(w, w') \in E_l\})$
  $E'_l \overset{\text{def}}{=} E'_l \cup (\{(w', v) : (w', w) \in E_l\} \cup \{(v, w') : (w, w') \in E_l\})$
  $E'_d \overset{\text{def}}{=} E_d \setminus (\{\{w, w'\} \in E_d\}) \cup (\{\{v, w'\} : \exists w'.\{w, w'\} \in E_d\})$
  $\ell'(x) \overset{\text{def}}{=} v$ if $\ell(x) = w$ and $\ell'(x) \overset{\text{def}}{=} \ell(x)$ otherwise
  $G' \overset{\text{def}}{=} (V'_b, V'_r, E'_p, E'_l, E'_d, \ell')$
  return $G'$
```

Lemma 6.2.4 *Let $\alpha(G) = (\phi; \sigma * x \mapsto y))$ $v, w \in V_{b,r}$, $x = \mathrm{var}(v)$ and $y = \mathrm{var}(w)$. We have $\alpha(\text{P}\textsc{Remove}(G, (v, w))) \equiv (\phi; \sigma)$.*

We introduce functions $\text{LRe}\textsc{Merge}(G, (v, w))$ and $\text{PRe}\textsc{Merge}(G, (v, w))$ as macro, which first remove an l- respectively p-edge (v, w) from G and then merge w into v.

Coming back to our original goal which was to show how to deal with spatial assertions, Algorithm $\textsc{Apply}(G, \sigma)$ takes an SL-graph G and a single spatial assertion $\sigma \in \{x \mapsto y, \ell s(x, y)\}$ as input and outputs an SL-graph G' such that if $\alpha(G) = (\phi; \sigma')$ then $\alpha(G') \equiv (\phi; \sigma' * \sigma)$. Some extra care has to be taken if an l-edge is added that is already present in G, since $(\phi; \sigma * \ell s(x, y) * \ell s(x, y)) \equiv (\phi \wedge x = y; \sigma * \ell s(x, y))$. It

Algorithm 5 PRemove(G, v, w) removing a p-edge (v, w) semantically

```
Input: An SL-graph $G$, a $p$-edge $(v, w) \in E_p$
  if $|E_p(v)| > 1$ then
    return $G[(E_p \setminus \{(v, w)\})/E_p]$
  else
    return $G[(V_r \setminus \{v\})/V_r, (V_b \cup \{v\})/V_b, (E_p \setminus \{(v, w)\})/E_p]$
  end if
```

is easily checked that Apply runs in polynomial time.

By combining the algorithms considered in this section and computing the SL-graph corresponding to an assertion α by induction on the structure of α, we obtain the following lemma.

Lemma 6.2.5 *Let α be an SL-graph. Then there exists a polynomial-time algorithm that computes an SL-graph G such that $\alpha \equiv \alpha(G)$.*

We now are now going to move towards defining the normal form of an SL-graph and show that any SL-graph can be transformed into one in normal form such that their corresponding assertions are equivalent. A key concept of the normal form is that of a persistent set of edges.

Definition 19 Let G be an SL-graph, a set of edges $E \subseteq E_{p,l}$ is *persistent* if $V(E) \cap V_r \neq \emptyset$ or there are $v, w \in V(E)$ such that $\{v, w\} \in E_d$. $\diamondsuit$

Let us illustrate this definition with the help of Figure 6.2. Let e_1 be the l-edge from x_4 to x_5 and e_2 the l-edge from x_4 to x_7 of graph (a) in Figure 6.2. Neither $\{e_1\}$ nor $\{e_2\}$ is persistent, but $\{e_1, e_2\}$ is as there is a d-edge between x_5 and x_7. Intuitively, the idea behind the definition is as follows: suppose we are given an SL-graph G with $(v, w) \in E_l$ such that $E = E^*_{p,l}(v, w)$ is persistent. Then in any model $\mathcal{I}$ of $\alpha(G)$ for $v' = \ell^{\mathcal{I}}(\text{var}(v))$, we have $v' \in V_r^{\mathcal{I}}$ since v' must have an outgoing p-edge as the persistence property enforces that there is a p-edge in E or that not all variable names occurring in E are mapped to v' in $\mathcal{I}$. Moreover, if v has a further outgoing l-edge (v, w') then $\ell^{\mathcal{I}}(\text{var}(w')) = v$ since v can only have one outgoing p-edge in $\mathcal{I}$.

Algorithm 6 $\textsc{Apply}(G, \sigma)$ adding a p or l-edge semantically

```
Input: An SL-graph $G$, an assertion $\sigma \in \{x \mapsto y, \ell s(x, y)\}$
  $v \overset{\text{def}}{=} \ell(x); w \overset{\text{def}}{=} \ell(y); \ell' \overset{\text{def}}{=} \ell; V_b' \overset{\text{def}}{=} V_b$
  if $\ell(x)$ is undefined then
    $v \overset{\text{def}}{=} \text{choose}(V \setminus V_{b,r}); V_b' \overset{\text{def}}{=} V_b' \cup \{v\}; \ell' \overset{\text{def}}{=} \ell'[x \mapsto v]$
  end if
  if $\ell(y)$ is undefined then
    $w \overset{\text{def}}{=} \text{choose}(V \setminus V_{b,r}); V_b' \overset{\text{def}}{=} V_b' \cup \{w\}; \ell' \overset{\text{def}}{=} \ell'[y \mapsto v]$
  end if
  if $\sigma = x \mapsto y$ then
    if $(\ell(x), \ell(y)) \in E_p$ then
      return $\perp$
    end if
    $V_r' \overset{\text{def}}{=} V_r \cup \{\ell(x)\}; V_b' = V_b' \setminus \{\ell(x)\}$
    $E_p' \overset{\text{def}}{=} E_p \cup \{(\ell(x), \ell(y)\}; G' \overset{\text{def}}{=} (V_r', V_b', E_p', E_l, E_d, \ell)$
  end if
  if $\sigma = \ell s(x, y)$ then
    if $(\ell(x), \ell(y)) \in E_l$ then
      $G' \overset{\text{def}}{=} \textsc{Merge}(G, \ell(x), \ell(y))$
    else
      $E_l' \overset{\text{def}}{=} E_l \cup \{\ell(x), \ell(y)\}; G' \overset{\text{def}}{=} (V_r, V_b', E_p, E_l', E_d, \ell)$
    end if
  end if
  return $G'$
```

(i) if $v \in V_r$ then $|E_p(v)| = 1$

(ii) if $v \rightarrow_{p,l} w$ such that $E^*_{p,l}(v, w)$ is persistent then $E_l(v) \subseteq \{w\}$

(iii) if $v \rightarrow_l w_1$ and $v \rightarrow_l w_2$ such that $E^*_{p,l}(v, w_1) \cup E^*_{p,l}(v, w_2)$ is persistent then $E_l(v) \subseteq \{w_1, w_2\}$

(iv) there are no distinct loop-free v-w paths π_1, π_2 in E_l.

Table 6.2: Conditions for an SL-graph G to be reduced.

For graph (a) in Figure 6.2, this means that x_6 becomes equivalent to x_4 in any model of the corresponding SL-formula. Thus persistency allows us to make some implicit equalities in G explicit.

We can now give a definition of our normal form of SL-graphs.

Definition 20 An SL-graph G is *reduced* if $G = \bot$ or if it fulfills the conditions in Table 6.2. An SL-graph G is in *normal form* if G is reduced and for all $v, w \in V_{b,r}$ such that $\alpha(G) = (\phi; \sigma)$, $x = \text{var}(v)$ and $y = \text{var}(w)$, whenever $(\phi \wedge x = y; \sigma)$ is unsatisfiable then $\{v, w\} \in E_d$.

Thus, an SL-graph is in normal form if it is reduced and if its set of d-edges is saturated. Let us explain on an informal level the four conditions given in Table 6.2 that constitute the property of a graph being reduced. The idea behind those conditions is that if any of them is violated by an SL-graph G then we can make some implicit facts explicit. Clearly, if (i) is violated then $\alpha(G)$ is unsatisfiable as the spatial part of $\alpha(G)$ consists of a statement of the form $x \mapsto y * x \mapsto z$. If (ii) or (iii) is violated then by our previous discussion on persistent edges any further outgoing l-edge can be collapsed into v. Condition (iv) contributes to making sure that between any two different nodes there is at most one loop-free path, as can be seen by the following lemma. Note that in particular any interpretation $\mathcal{I}$ is an SL-graph in normal form.

Algorithm 7 REDUCE(G) reducing G according to Definition 20

```
Input: An SL-graph $G$
  while $G$ is not reduced do
    case split on violated condition at node $v$
    // conditions are as in Table 6.2
    // node names below refer in each case to the corresponding case in Lemma 6.2.9

    case (i): return $\bot$
    case (ii): $G = \text{LReMerge}(G, (v, w'))$
    case (iii): $G = \text{LReMerge}(G, (v, w''))$
    case (iv): $G = \text{Merge}(G', v, w)$
  end while
  return $G$
```

Lemma 6.2.6 *Let $G \neq \bot$ be a reduced SL-graph, v, w be distinct nodes in $V_{b,r}$ and $\pi : v \rightsquigarrow_{l,r} w$ a loop-free path. Then π is the unique such loop-free path.*

Proof. To the contrary, assume that there are two different loop-free v-w paths π_1, π_2. Then there are nodes v', w' such that there are distinct v'-w' paths π'_1 and π'_2 that are segments of π_1 respectively π_2, where at least one of π_1 or π_2 is of non-zero length. If $v' = w'$ then this contradicts to π_1 or π_2 being loop-free. Thus, assume $v' \neq w'$. If both π'_1, π'_2 are l-paths then this contradicts to G being reduced, as condition (iv) is violated. Otherwise, if π'_1 reaches a red node then edges(π'_1) is persistent and hence v' has one outgoing edge, contradicting to π'_1 and π'_2 being distinct. The case when π'_2 reaches a red node is symmetric. □

It is easy to see that checking whether a graph G is reduced can be decided in polynomial time in $|G|$. In order to transform an arbitrary SL-graph into a reduced SL-graph, Algorithm REDUCE(G) just checks for a given input G if any condition from Table 6.2 is violated. If this is the case, the algorithm removes edges and merges nodes, depending on which condition is violated, until G is reduced. We will subsequently prove REDUCE to be correct. First, we provide two technical lemmas that will help us to prove correctness. They allow us to formalise our intuition about persistent sets of edges.

Lemma 6.2.7 *Let G be an SL-graph and $v, w, w' \in V_{b,r}$ such that $x = \mathrm{var}(v)$, $y = \mathrm{var}(w)$, $v \rightsquigarrow_l w$, and let $\mathcal{I}$ be a model of $\alpha(G)$. Then the following holds:*

(i) if $\ell^{\mathcal{I}}(y) \in V_r^{\mathcal{I}}$ then $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$; and

(ii) if $v \rightsquigarrow_l w'$ and $\{w, w'\} \in E_d$ then $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$.

Proof. (i) Let $\alpha(G) = (\phi; \sigma)$. The proof is by induction on the length of the l-path π from v to w. The case $|\pi| = 0$ is trivial. For the induction step, let $\pi = vw' \cdot \pi'$. Then $\ell s(x, z)$ is a spatial assertion in σ with $z = var(w')$. By the induction hypothesis, $\ell^{\mathcal{I}}(z) \in V_r^{\mathcal{I}}$. Moreover, $\mathcal{I} \models \ell s(x, z)$ and hence $\mathcal{I} \models \ell s^n(x', z)$ for some $n \in \mathbb{N}$. If $n = 0$ we have $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(z) \in V_r$. Otherwise, $\mathcal{I} \models x \mapsto z' * \ell s^{n-1}(z', z)$, where $z' \in \mathit{Vars}$ is fresh. Consequently, $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$.

(ii) Let $\alpha(G) = (\phi; \sigma)$ and π and π' be l-paths from v to w respectively w'. We show the statement by induction on $|\pi| + |\pi'|$. Let $m = |\pi| + |\pi'|$. As $\{w, w'\} \in E_d$, we have $m > 0$. If $m = 1$, assume without loss of generality that $|\pi| = 1$ and let $y = var(w)$. For any model $\mathcal{I}$ of $\alpha(G)$, $\mathcal{I} \models \ell s(x, y)$ and hence $\mathcal{I} \models \ell s^n(x, y)$ for some $n \in \mathbb{N}$. Since $\mathcal{I} \models x \neq y$, $n > 0$ and thus $\mathcal{I} \models x \mapsto z * \ell s^{n-1}(z, y)$ for some fresh $z \in \mathit{Vars}$. Consequently, $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$. For $m > 1$, we assume without loss of generality that $|\pi| > 0$, i.e., $\pi = vu \cdot \pi''$. Hence $\ell s(x, y)$ is a spatial assertion in σ, where $y = var(u)$. We have $\mathcal{I} \models \ell s(x, y)$ and hence $\mathcal{I} \models \ell s^n(x, y)$ for some $n \in \mathbb{N}$. If $n > 0$ then $\mathcal{I} \models x \mapsto z * \ell s^{n-1}(z, y)$ for some fresh $z \in \mathit{Vars}$, hence $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$. If $n = 0$, let $G' \overset{\text{def}}{=} \textsc{RLMerge}(G, (v, u))$. As $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$, it follows that $\mathcal{I}$ is a model of $\alpha(G')$, hence the induction hypothesis yields that $\ell^{\mathcal{I}}(x) \in V_r$. □

The next lemma formalises our intuition about l-edges whose source node is guaranteed to be red in any model. It shows that in this case any outgoing l-edge collapses into its source node.

Lemma 6.2.8 *Let $\alpha = (\phi, \sigma)$ and $x \in$ Vars be such that for all models $\mathcal{I}$ of α, $\ell^{\mathcal{I}}(x) \in V_r^{\mathcal{I}}$. Then for all $y \in$ Vars and $\alpha' = (\phi, \sigma * \ell s(x, y)), \alpha'' = (\phi \wedge x = y, \sigma)$, we have $\alpha' \equiv \alpha''$.*

Proof. We clearly have that $\alpha'' \models \alpha'$. For the other direction, let $\mathcal{I}'$ be a model of α'. By definition, there are $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathcal{I}' = \mathcal{I}_1 * \mathcal{I}_2$, $\mathcal{I}_1 \models (\phi; \sigma)$ and $\mathcal{I}_2 \models (\phi; \ell s(x, y))$. By assumption, $\ell^{\mathcal{I}_1}(x) \in V_r^{\mathcal{I}_1}$ and hence $\ell^{\mathcal{I}_2}(x) \notin V_r^{\mathcal{I}_2}$. Consequently, $\ell^{\mathcal{I}_2}(x) = \ell^{\mathcal{I}_2}(y)$. Hence $\ell^{\mathcal{I}'}(x) = \ell^{\mathcal{I}'}(y)$, which yields $\mathcal{I}' \models (\phi \wedge x = y; \sigma)$. □

We are now prepared to show the correctness of Reduce. Each case in the lemma below captures a violated condition from Table 6.2 and shows that the manipulation performed by Reduce is sound and complete.

Lemma 6.2.9 *Let G be an SL-graph,*

(i) if there is $v \in V_r$ such that $|E_p(v)| > 1$ then $\alpha(G)$ is unsatisfiable;

(ii) if there are $v, w, w' \in V_{b,r}$, $x, y \in$ Vars such that $v \to_{p,l} w$, $v \to_l w'$, $x = \mathrm{var}(v)$, $y = \mathrm{var}(w')$, $E^*_{p,l}(v, w)$ is persistent and $\alpha(G) = (\phi, \sigma * \ell s(x, y))$ then $\alpha(G) \equiv (\phi \wedge x = y; \sigma)$;

(iii) if there are $v, w, w', w'' \in V_{b,r}$, $x, y \in$ Vars such that $v \to_l w$, $v \to_l w'$, $v \to_l w''$, $x = \mathrm{var}(v)$, $y = \mathrm{var}(w'')$, $E^*_{p,l}(v, w) \cup E^*_{p,l}(v, w')$ is persistent and $\alpha(G) = (\phi, \sigma * \ell s(x, y))$ then $\alpha(G) \equiv (\phi \wedge x = y; \sigma)$; and

(iv) if there are $v, w \in V_b$, $x, y \in$ Vars such that $x = \mathrm{var}(v)$, $y = \mathrm{var}(w)$, $\alpha(G) = (\phi, \sigma)$ and there are distinct loop-free v-w l-paths π_1, π_2 in E_l then $\alpha(G) \equiv (\phi \wedge x = y; \sigma)$.

Proof. Case (i): Let $x = \mathrm{var}(v)$, we have that there are $y, z \in$ *Vars* such that $(\phi; \sigma * x \mapsto y * x \mapsto z)$, which clearly is unsatisfiable.

Case (ii): We show that for all models $\mathcal{I}$ of $\alpha(G)$, $\ell^{\mathcal{I}}(x) \in V_r$. The statement then follows from Lemma 6.2.8. If there is $u \in V(E^*_{p,l}(v, w)) \cap V_r$ then by Lemma 6.2.7(i) we have $x \in V_r^{\mathcal{I}}$. Otherwise, if there are $u, u' \in V(E^*_{p,l}(v,w))$ such that $\{u,u'\} \in E_d$ then Lemma 6.2.7(ii) gives $x \in V_r^{\mathcal{I}}$.

Case (iii): Again, we show that for all models $\mathcal{I}$ of $\alpha(G)$, $\ell^{\mathcal{I}}(x) \in V_r$. The statement then follows from Lemma 6.2.8. It is sufficient to consider the case in which there are $u, u' \in V_{b,r}$ such that $w \rightsquigarrow_l u$, $w \rightsquigarrow_l u'$ and $\{u,u'\} \in E_d$ as all other cases are subsumed by (ii). But then, Lemma 6.2.7(ii) again yields $x \in V_{b,r}^{\mathcal{I}}$.

Case (iv): Let $\pi_1 = vw_1 \cdot \pi_1'$ and $\pi_2 = vw_2 \cdot \pi_2'$ be v-w paths. Thus, $w_1 \neq w_2$ and hence $m \overset{\text{def}}{=} |\pi_1| + |\pi_2| \geq 3$. We show the statement by induction on m. For $m = 3$, the statement follows from a similar reasoning as in Lemma 6.2.8. For the induction step, let $m > 3$ and $\mathcal{I}$ be model of $\alpha(G)$. Let $y_1 = \text{var}(w_1)$ and $y_2 = \text{var}(w_2)$, we have that $\alpha(G) = (\phi; \sigma * \ell s(x,y_1) * \ell s(x,y_2))$ and consequently $\mathcal{I} \models \sigma * \ell s^{n_1}(x,y_1) * \ell s^{n_2}(x,y_2)$ for some $n_1, n_2 \in \mathbb{N}$. If $n_1 = 0$ then $\mathcal{I} \models G'$, where $G' = \textsc{LReMerge}(G,(v,w_1))$ and the induction hypothesis yields $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$. The case $n_2 = 0$ follows symmetrically. □

Proposition 6.2.1 *Let G, G' be SL-graphs such that $G' = \textsc{Reduce}(G)$. Then G' is reduced and $\alpha(G) \equiv \alpha(G')$. Moreover,* Reduce *runs in polynomial time on any input G.*

Proof. Clearly, Reduce only returns graphs that are reduced. Moreover, Lemma 6.2.9 shows that in every iteration equivalent graphs are generated and hence $\alpha(G) \equiv \alpha(G')$. Regarding the complexity, checking if G is reduced can be performed in polynomial time in $|G|$. Removing edges and merging nodes in the **while**-body can also be performed in polynomial time. Moreover, the size of G strictly decreases after each iteration of the **while**-body. Hence the **while**-body is only executed a polynomial number of times. □

The next proposition summarises all constructions that we have established in this section so far.

Proposition 6.2.2 *For any SL-formula α, there exists a polynomial time computable SL-graph G in normal form such that $\alpha \equiv \alpha(G)$.*

Proof. Given an assertion $\alpha = (\phi; \sigma)$, by Lemma 6.2.5 we can construct an SL-graph G' such that $\alpha(G') \equiv \alpha$. Applying REDUCE to G' yields a reduced graph G'' such that $\alpha(G') \equiv \alpha(G'')$. In order to bring G'' into normal form, we check for each of the polynomially many pairs $v, w \in V_{b,r}$ if REDUCE returns $\bot$ on input MERGE(G'', v, w). If this is the case, we add $\{v, w\}$ to E_d, which finally gives us the desired graph G. As argued before, all constructions can be performed in polynomial time. □

Let us now illustrate our definitions with the help of an example. Graph (b) in Figure 6.2 is in normal form and obtained from the graph (a) by applying REDUCE. Graph (a) violates condition (iii) of Table 6.2 as $\{(\ell(x_4), \ell(x_5)), (\ell(x_4), \ell(x_7)\}$ is persistent, which results in REDUCE merging x_6 into x_4. Moreover, the graph also violates condition (iv) since there are two distinct l-paths from x_1 to x_3. Hence, REDUCE merges x_1 and x_3 and then removes all newly obtained outgoing l-edges from x_3 due to a violation of condition (ii). Finally in order to obtain graph (b) in normal form, $\{(\ell(x_3), \ell(x_4))\}$ is added to E_d as merging the nodes x_3 and x_4 and applying REDUCE results in an inconsistent graph.

As stated before, a nice property of SL-graphs in normal form is that they allow to easily construct a model of their corresponding SL-formulae.

Lemma 6.2.10 *Let $G \neq \bot$ be a reduced SL-graph and $v, w \in V_{b,r}$ such that $v \neq w$. Then $\alpha(G)$ has a model $\mathcal{I}$ such that $\ell^{\mathcal{I}}(\text{var}(v)) \neq \ell^{\mathcal{I}}(\text{var}(w))$ and for all $x, y \in Vars$, $\ell(x) = \ell(y)$ implies $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$.*

Proof. We sketch how G can iteratively be turned into a desired model $\mathcal{I}$. Suppose w is reachable from v and let π be the loop-free path from v to w. First, we replace any l-edge occurring on π by *two* consecutive p-edges. For all nodes $v' \neq w$ along π that have further outgoing l-edges, we merge all nodes reachable via l-paths from v' into v' and remove the connecting l-edges. If v is reachable from w via a loop-free path π', we apply the same procedure to π'. Finally, we iterate the following procedure: if there is a node u with more than one outgoing l-edge, we fix an l-edge e and merge all nodes reachable from u via the remaining l-edges different from e into u and remove the connecting l-edges. We then replace e with two new consecutive p-edges. Once this procedure has finished, we obtain an SL-graph containing no l-edges that can be turned into an interpretation $\mathcal{I}$. It is easily checked that $\mathcal{I}$ is a model of $\alpha(G)$ and $\ell^{\mathcal{I}}(\mathrm{var}(v)) \neq \ell^{\mathcal{I}}(\mathrm{var}(w))$. □

As the reduced SL-graph corresponding to an assertion can be computed in polynomial time, we have thus shown that checking satisfiability in our assertion language is decidable in polynomial time.

Theorem 6.2.1 *Satisfiability of SL-formulae is decidable in polynomial time.*

6.2.2 SL-Graph Homomorphisms

In this section, we are going to show that entailment between SL-formulae can be decided by checking for the existence of a graph homomorphism between their corresponding SL-graphs in normal form. Throughout this section, we will assume that all SL-formulae considered are satisfiable and all SL-graphs $G \neq \bot$, since deciding entailment becomes trivial otherwise, and checking for satisfiability can be done in polynomial time.

A homomorphism is a mapping between the nodes of two SL-graphs that, if it exists, preserves the structure of the source graph in the target graph. In the definition of a homomorphism, we make use of the property of SL-graphs in normal form that between any disjoint nodes there is at most one loop-free path connecting the two nodes, *c.f.* Lemma 6.2.6. For nodes $v \neq w$, we denote this path by $\pi(v, w)$ if it exists. If $v = w$ then $\pi(v, w)$ is the zero-length path $\pi(v, w) \overset{\text{def}}{=} v$. Subsequently, let us fixed SL-graphs $G = (V_b, V_r, E_l, E_p, E_d, \ell)$, $G' = (V_b', V_r', E_l', E_p', E_d', \ell')$ and $''G = (V_b'', V_r'', E_l'', E_p'', E_d'', \ell'')$.

Definition 21 Let G, G' be SL-graphs in normal form. A mapping $h : V_{b,r} \to V'_{b,r}$ is a *homomorphism* from G to G' if the *homomorphism conditions* from Table 6.3 are satisfied. ◊

Given a mapping h, it is easy to see that checking whether h is a homomorphism can be performed in polynomial time in $|G| + |G'|$. The goal of this section is to prove the

(i) $\mathrm{vars}(v) \subseteq \mathrm{vars}(h(v))$

(ii) if $\{v, w\} \in E_d$ then $\{h(v), h(w)\} \in E'_d$

(iii) if $v \rightarrow_p w$ then $h(v) \rightarrow'_p h(w))$

(iv) if $v \rightarrow_l w$ then $h(v) \rightsquigarrow'_{p,l} h(w)$

(v) for all $v_1 \rightarrow_{p,l} w_1$ and $v_2 \rightarrow_{p,l} w_2$ such that $(v_1, w_1) \neq (v_2, w_2)$, $\mathrm{edges}(\pi(h(v_1), h(w_1))) \cap \mathrm{edges}(\pi(h(v_2), h(w_2))) = \emptyset$

(vi) if $v, w \in V_r$ and $v \neq w$ then $h(v) \neq h(w)$

Table 6.3: Conditions for a homomorphism h from G to G'.

following proposition, which gives us the relationship between homomorphisms and entailment.

Proposition 6.2.3 *Let G, G' be SL-graphs in normal form. Then $\alpha(G') \models \alpha(G)$ if, and only if, there exists a homomorphism h from G to G'.*

Before we begin with formally proving the proposition, let us discuss its validity on an intuitive level. Suppose there is a homomorphism from G to G'. Condition (i) makes sure that for any node v of G its image under h is labelled with at least the same variables. If this were not the case, we could easily construct a counter-model of $\alpha(G')$ disproving entailment. Likewise, condition (ii) ensures that whenever two nodes are required to be not equivalent, the same is true for the two nodes under the image of h. Since G' is in normal form, merging the two nodes in the image of h would otherwise be possible since E'_d is maximal. Condition (iii) requires that whenever there is a p-edge between any two nodes v, w, such an edge also exists in G'. Again, it is clear that if this were not the case we could construct a counter-model $\mathcal{I}$ of $\alpha(G')$ such that there is no p-edge between $\ell^{\mathcal{I}}(\mathrm{var}(v))$ and $\ell^{\mathcal{I}}(\mathrm{var}(w))$. Condition (iv) is of a similar nature, but here we allow that there is a whole path between $h(v)$ and $h(w)$. In condition (v), we require that the paths obtained from the image of two disjoint edges do not share a common edge in G'. If this were the case, we could construct a model of $\alpha(G')$ in which separation is violated. Finally, condition (vi) makes sure that no two different nodes from V_r are mapped to the same node. This condition is needed to handle p-edges of the form (v, v), which may not be covered by condition (v). We now proceed with formally proving Proposition 6.2.3. First, the following lemma shows the relationship between models and homomorphisms and that homomorphisms can be composed.

Lemma 6.2.11 *Let G, G', G'' be SL-graphs in normal form and $\mathcal{I}$ an interpretation. Then the following holds:*

(i) let $h : V_{b,r} \to V^{\mathcal{I}}_{b,r}$ be such that for all $v \in V_{b,r}$, $h(v) \overset{def}{=} \ell^{\mathcal{I}}(\mathrm{var}(v))$; then $\mathcal{I} \models \alpha(G)$ if, and only if, h is a homomorphism from G to $\mathcal{I}$; and

(ii) given homomorphisms h', h'' from G' to G respectively G'' to G'; then $h \overset{def}{=} h'' \circ h'$ is a homomorphism from G'' to G.

Proof. (i) Throughout this proof, we make implicit use of the fact that for all $\mathcal{I}$, $\mathcal{I} \models (\phi; \sigma * \sigma')$ if, and only if, there are $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$, $\mathcal{I}_1 \models (\phi; \sigma)$ and $\mathcal{I}_2 \models (\phi; \sigma')$.

("$\Leftarrow$") We show the statement by induction on $|\sigma|$. In the following, let $\alpha(G) = (\phi, \sigma)$. It is not difficult to check that the statement holds in the induction base case.

For the induction step, let us first consider the case $\alpha(G) = (\phi; \sigma' * x \mapsto y)$. We show that there are $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathcal{I}_1 \models (\phi; \sigma')$, $\mathcal{I}_2 \models (\phi; x \mapsto y)$ and $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$. Let $v = \ell(x)$ and $w = \ell(y)$. Set $\mathcal{I}_2 \overset{\text{def}}{=} \mathcal{I}[\{h(v)\}/V_r, V_b \cup (V_r \setminus \{h(v)\})/V_b, \{(h(v), h(w))\}/E_p]$. Clearly, $\mathcal{I}_2 \models x \mapsto y$. Choose $\mathcal{I}_1$ such that $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$ and let $G' = \textsc{PRemove}(G, (v, w))$ be an SL-graph in normal form such that $\alpha(G') = (\phi; \sigma')$. It is not difficult to verify that h is a homomorphism from G' to $\mathcal{I}_1$. In particular conditions (iii) and (iv) are satisfied by h since condition (v) makes sure that when we remove the p-edge $(h(v), h(w))$ from $\mathcal{I}$ then we do not destroy any other path under the image of h. It follows from the induction hypothesis that $\mathcal{I}_1 \models (\phi; \sigma')$.

Next, suppose $\alpha(G) = (\phi; \sigma' * \ell s(x, y))$. Again, we show that there are $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathcal{I}_1 \models (\phi; \sigma')$, $\mathcal{I}_2 \models (\phi; \ell s(x, y))$ and $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$. Let $v = \ell(x)$ and $w = \ell(y)$ and $\pi(h(v), h(w)) = v'_1 v'_2 \cdots v'_n$. Define $V'_r \stackrel{\text{def}}{=} \{v'_1, v'_2, \ldots, v'_{n-1}\}$, which is possibly an empty set when $\pi(h(v), h(w))$ has length zero. Now set $\mathcal{I}_2 = \mathcal{I}[V'_r/V^{\mathcal{I}}_r, V_b \cup (V^{\mathcal{I}}_r \setminus V'_r)/V^{\mathcal{I}}_b, \{(v'_i, v'_{i+1}) : 1 \leq i < n\}/E^{\mathcal{I}}_p]$ and choose $\mathcal{I}_1$ such that $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$. Let $G' \stackrel{\text{def}}{=}$ LRemove$(G, (v, w))$, it follows that G' is an SL-graph G' in normal form such that $\alpha(G') = (\phi; \sigma')$ and h is a homomorphism from G' to $\mathcal{I}_1$. The induction hypothesis then yields $\mathcal{I}_1 \models \alpha(G')$ as required.

("$\Rightarrow$") Let $\alpha(G) = (\phi; \sigma)$ and let $\mathcal{I}$ be a model of $\alpha(G)$. We show by the statement by induction on the number of spatial assertions in σ. The induction base case is reasonably clear, in particular $\{\ell^{\mathcal{I}}(x), \ell^{\mathcal{I}}(y)\} \in E^{\mathcal{I}}_d$ if, and only if, $\ell^{\mathcal{I}}(x) \neq \ell^{\mathcal{I}}(y)$ ensures that condition (ii) is satisfied. For the induction step, let us consider the case $\alpha(G) = (\phi; \sigma' * x \mapsto y)$, the case $\alpha(G) = (\phi; \sigma' * \ell s(x, y))$ follows along similar lines. Let $v = \ell(x)$, $w = \ell(y)$ and $G' =$ PRemove$(G, (v, w))$, so that $\alpha(G') = (\phi; \sigma)$. As $\mathcal{I} \models \alpha(G)$, we have $\mathcal{I} = \mathcal{I}' * \mathcal{I}''$ such that $\mathcal{I}' \models (\phi; \sigma')$ and $\mathcal{I}'' \models (\phi; x \mapsto y)$. By the induction hypothesis, there exists a homomorphism h' from G' to $\mathcal{I}'$. Set $h \stackrel{\text{def}}{=} h'[\ell(x) \mapsto \ell^{\mathcal{I}}(x), \ell(y) \mapsto \ell^{\mathcal{I}}(y)]$, we claim that h is a homomorphism from G to $\mathcal{I}$. Conditions (i), (ii) and (iv) are obviously satisfied. For (iii), since $\mathcal{I}'' \models x \mapsto y$, we have that $(\ell^{\mathcal{I}}(x), \ell^{\mathcal{I}}(y)) \in E^{\mathcal{I}''}_p \subseteq E^{\mathcal{I}}_p$ and since h' is a homomorphism, condition (iii) is true for all remaining p-edges that are in G' as well. Regarding (v), if both (v_1, w_1) and (v_2, w_2) are edges in G' then condition (v) holds, since h' is a homomorphism. Otherwise, suppose $(v_1, w_1) = (\ell(x), \ell(y)) \in E_p$, then by the semantics definition there is no p-edge in $\mathcal{I}'$. Since (iii) and (iv) hold for h', $\{(\ell(x), \ell(y))\} \cap \text{edges}(\pi(h(v_2), h(w_2)))$ must be empty. Last, it is easily verified that (vi) holds as well.

(ii) We check that all homomorphism conditions are met. Regarding condition (i), let $v \in V_{b,r}$. We have $\text{vars}(v) \subseteq \text{vars}(h'(v)) \subseteq \text{vars}(h''(h'(v))) = \text{vars}(h(v))$. For (ii), from h' we have $(v, w) \in E''_d$ implies $(h'(v), h'(w)) \in E'_d$ and hence h'' implies $(h''(h''(v)), h''(h'(w))) \in E''_d$, *i.e.*, $(h(v), h(w)) \in E''_d$. For (iii), for $v \rightarrow_p w$, h' gives $h'(v) \rightarrow'_p h'(w)$ and h'' yields $h''(h'(v)) \rightarrow''_p h''(h'(v))$, *i.e.*, $h(v) \rightarrow''_p h(w)$. Likewise for (iv), if $v \rightarrow_l w$ then h' gives $h'(v) \rightsquigarrow'_{p,l} h'(w)$. Now for any p- or l-edge (v', w') along the path $\pi' = \pi(h'(v), h'(w))$, h'' yields $h''(v') \rightsquigarrow''_{p,l} h''(w')$, which implies $h(v) \rightsquigarrow''_{p,l} h(w)$. We now show that (v) holds for h. To this end, let $v_1 \rightarrow_{p,l} w_1$ and $v_2 \rightarrow_{p,l} w_2$. Let $\pi'_1 = \pi(h(v_1), h(w_1))$ and $\pi'_2 = \pi(h(v_2), h(w_2))$, from h' we get $\text{edges}(\pi'_1) \cap \text{edges}(\pi'_2) = \emptyset$. Now h'' gives us that for any pair of edges (v'_1, w'_1) and (v'_2, w'_2) that appear along π'_1 respectively π'_2, $\text{edges}(\pi''_1) \cap \text{edges}(\pi''_2) = \emptyset$, where $\pi''_1 = \pi(h''(v'_1), h''(w'_1))$ and $\pi''_2 = \pi(h''(v'_2), h''(w'_2))$. Hence, $\text{edges}(\pi(h(v_1), h(w_1))) \cap \text{edges}(\pi(h(v_2), h(w_2))) = \emptyset$. Finally for (vi), let $v, w \in V_{b,r}$ such that $v \neq w$. Then $h'(v) \neq h'(w)$ and $h'(v), h'(w) \in V'_{b,r}$ and hence $h(v) = h''(h'(v)) \neq h''(h'(w)) = h(w)$. $\square$

Proposition 6.2.3 now is a consequence of the following lemma. Note that the homomorphism is fully determined by G and G'.

Lemma 6.2.12 *Let G, G' be SL-graphs in normal form and let $h : V_{b,r} \rightarrow V'_{b,r}$ be defined as $h(v) \overset{def}{=} \ell'(\text{var}(v))$ for all $v \in V_{b,r}$. Then $\alpha(G') \models \alpha(G)$ if, and only if, h is a homomorphism from G to G'.*

Proof. ("$\Leftarrow$") Let h be a homomorphism from G to G' and $\mathcal{I}$ be such that $\mathcal{I} \models \alpha(G')$. By Lemma 6.2.11(i), there exists a homomorphism h' from G' to $\mathcal{I}$. By Lemma 6.2.11(ii), $h'' \overset{\text{def}}{=} h' \circ h$ is a homomorphism from G to $\mathcal{I}$. Consequently, Lemma 6.2.11(i) yields $\mathcal{I} \models \alpha(G)$.

("$\Rightarrow$") Let $\alpha(G) = (\phi, \sigma)$ and $\alpha(G') = (\phi', \sigma')$. We show the contrapositive. Suppose h is not a homomorphism from G to G', we construct a counter-model $\mathcal{I}$ such that $\mathcal{I} \models \alpha(G')$ and $\mathcal{I} \not\models \alpha(G)$.

Suppose condition (i) is violated by h. Then there are $x, y \in \mathit{Vars}$ such that $\ell(x) = \ell(y) = v$ and $\ell'(x) = v' \neq w' = \ell(y)$. By Lemma, $\alpha(G')$ then has a model $\mathcal{I}$ such that $\ell^{\mathcal{I}}(v') \neq \ell^{\mathcal{I}}(w')$, which clearly is not a model of $\alpha(G)$.

Next, suppose condition (ii) is violated by h. Then there are $\{v, w\} \in E_d$ such that $\{h(v), h(w)\} \notin E'_d$. Let $x = \text{var}(h(v))$ and $y = \text{var}(h(w))$, since G' is in normal form $(\phi' \wedge x = y; \sigma')$ is satisfiable, *i.e.*, there exists $\mathcal{I}$ such that $\mathcal{I} \models \alpha(G')$ and $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$.

If condition (iii) is violated, let $(v, w) \in E_p$ such that $(h(v), h(w)) \notin E_{p,l}$ and let $\mathcal{I}$ be a model of $\alpha(G')$. If $(\ell^{\mathcal{I}}(var(h(v))), \ell^{\mathcal{I}}(var(h(w))) \notin E^{\mathcal{I}}_p$ we have the desired counter model. Otherwise, let h' be a homomorphism from G' to $\mathcal{I}$ and let $\mathcal{I}' \overset{\text{def}}{=}$ PRemove$(\mathcal{I}, (h(v), h(w)))$. Clearly, h' still is a homomorphism from G' to $\mathcal{I}'$, hence $\mathcal{I}' \models \alpha(G')$. However, obviously $\mathcal{I}' \not\models \alpha(G)$. If $(h(v), h(w)) \in E_l$ this approach does not work. But then Lemma 6.2.10 gives us a model $\mathcal{I}$ of $\alpha(G')$ in which this list is expanded to a path of length two, hence $(\ell^{\mathcal{I}}(\text{var}(v)), \ell^{\mathcal{I}}(\text{var}(w))) \notin E_p^{\mathcal{I}}$.

If condition (iv) is violated by h, we proceed along similar lines. Let $\mathcal{I}$ be such that $\mathcal{I} \models \alpha(G')$. If there is no path between $\ell^{\mathcal{I}}(\text{var}(h(v)))$ and $\ell^{\mathcal{I}}(\text{var}(h(w)))$ in $\mathcal{I}$ then $\mathcal{I}$ serves as the desired counter-model. Otherwise, let h' be a homomorphism from $\alpha(G')$ to $\mathcal{I}$. There is some edge $(v'', w'') \in E_p^{\mathcal{I}} \cap \text{edges}(\pi(\ell^{\mathcal{I}}(\text{var}(v), \ell^{\mathcal{I}}(\text{var}(w)))$ that is not in the image of h'. Define $\mathcal{I}' \overset{\text{def}}{=}$ PRemove$(G', (v'', w''))$. We clearly have that h' is a homomorphism from G' to $\mathcal{I}'$ and hence $\mathcal{I}' \models \alpha(G')$. However as there is no path between $\ell^{\mathcal{I}'}(\text{var}(v))$ and $\ell^{\mathcal{I}'}(\text{var}(w))$ anymore in $\mathcal{I}'$, $\mathcal{I}' \not\models \alpha(G)$.

Last, suppose (v) does not hold for h. Thus, there are $v_1, v_2, w_1, w_2 \in V_{b,r}$ and $v, w \in V'_{b,r}$ such that $(v_1, w_1), (v_2, w_2) \in E_{p,l}$ and $(v, w) \in \text{edges}(\pi(h(v_1), h(w_1))) \cap \text{edges}(\pi(h(v_2), h(w_2)))$. For simplicity, we assume $(v_1, w_1), (v_2, w_2) \in E_l$, the other cases following similarly. Moreover, let $x = \text{var}(v)$, $y = \text{var}(w)$, $x_i = \text{var}(v_i)$ and $y_i = \text{var}(w_i)$, $i \in \{1, 2\}$. By Lemma 6.2.10, $\alpha(G')$ has a model $\mathcal{I}$ such that $\ell^{\mathcal{I}}(x) \neq \ell^{\mathcal{I}}(y)$. Now for any separation of $\mathcal{I} = \mathcal{I}' * \mathcal{I}''$, only one of $\mathcal{I}'$ and $\mathcal{I}''$ can contain a path of length greater zero between $\ell^{\mathcal{I}}(x)$ and $\ell^{\mathcal{I}}(y)$, and hence in either $\mathcal{I}'$ or $\mathcal{I}''$ there is no path from $\ell^{\mathcal{I}}(x_1)$ to $\ell^{\mathcal{I}}(y_1)$ respectively $\ell^{\mathcal{I}}(x_2)$ to $\ell^{\mathcal{I}}(y_2)$. Hence, $\mathcal{I} \not\models \ell s(x_1, y_1) * \ell s(x_2, y_2)$ and consequently $\mathcal{I} \not\models \alpha(G)$.

In the last case (vi), we have that there exists a model $\mathcal{I}$ of $\alpha(G)$ such that $\ell^{\mathcal{I}}(x) \neq \ell^{\mathcal{I}}(y)$, where $x = \text{var}(v)$ and $y = \text{var}(w)$. Consequently, $\mathcal{I} \not\models x \mapsto z * y \mapsto z'$ for any $z, z' \in \mathit{Vars}$. Hence $\mathcal{I} \not\models \alpha(G)$. □

We can now combine all results of this chapter so far. Given satisfiable SL-formulae α and α', by Proposition 6.2.2 we can compute in polynomial time SL-graphs G and G' in normal form such that $\alpha \equiv \alpha(G)$ and $\alpha' \equiv \alpha(G')$. Next, we can compute in polynomial time a mapping h from $\alpha(G')$ to $\alpha(G)$ and check in polynomial time whether h is a homomorphism. By the previous lemma, this then is the case if, and only if, $\alpha \models \alpha'$.

Theorem 6.2.2 *Entailment between SL-formulae is decidable in polynomial time.*

An example of a homomorphism can be found in Figure 6.2. The arrows from graph (c) to graph (b) depict a homomorphism witnessing an entailment between the corresponding formulae of the graphs.

6.3 Syntactic Extensions Leading to Intractability

As stated in Section 6.2, due to the non-convexity present in our assertion language, it is rather surprising that entailment in our fragment is decidable in polynomial time. In this section, we briefly discuss natural syntactic extensions that render satisfiability or entailment intractable. It turns out that even small extensions make computing entailment intractable.

First, we consider additional Boolean connectives in pure and spatial formulae. Formally, we amend the syntax of pure formulae to

$$\phi ::= \mathit{expr} = \mathit{expr} \mid \neg\varphi \mid \varphi \wedge \varphi,$$

where $\mathcal{I} \models \neg\varphi$ if, and only if, $\mathcal{I} \not\models \varphi$, as expected. Clearly, we can reduce satisfiability of Boolean formulae to satisfiability in the extend assertion language. Since an assertion α is satisfiable if, and only if, $\alpha \not\models (x \neq x; \epsilon)$, we thus get that in the extended assertion language satisfiability is NP- and entailment coNP-hard.

It is not too surprising that allowing for all Boolean connectives in pure formulae makes satisfiability and entailment computationally hard. Less obvious, allowing for conjunction in spatial assertions makes satisfiability NP-hard and thus entailment coNP-hard. Formally, we amend the definition of the syntax of spatial formulae to

$$\sigma ::= \mathit{expr} \mapsto \mathit{expr} \mid \ell s(x, y) \mid \sigma * \sigma \mid \sigma \wedge \sigma,$$

where $\mathcal{I} \models \sigma_1 \wedge \sigma_2$ if, and only if, $\mathcal{I} \models \sigma_1$ and $\mathcal{I} \models \sigma_2$. In order to show our hardness results, we reduce from three colorability of undirected graphs.

3-Col

INPUT: An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

QUESTION: Does there exist a coloring $f : \mathcal{V} \to \{1, 2, 3\}$ such that $f(v) \neq f(w)$ whenever $(v, w) \in \mathcal{E}$?

3-Col is known to be NP-complete [45]. For our reduction, given an instance $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of 3Col with $\mathcal{V} = \{v_1, \ldots, v_n\}$, we construct an assertion α such that there exists a three-coloring of $\mathcal{G}$ if, and only if, α is satisfiable. We set $\alpha \stackrel{\text{def}}{=} (\phi, \sigma)$, where

$$\phi \stackrel{\text{def}}{=} \bigwedge_{(v_i,v_j)\in\mathcal{E}} x_i \neq x_j$$

$$\sigma \stackrel{\text{def}}{=} y_1 \mapsto y_2 * y_2 \mapsto y_3 \wedge \bigwedge_{v_i\in\mathcal{V}} \ell s(y_1, x_i) * \ell s(x_i, y_3).$$

Let us sketch the correctness of our reduction. The first conjunct of σ ensures that any model of α contains a list of three nodes that are successively labelled with the variable names y_1, y_2 and y_3. The remaining conjuncts enforce that for any $v_i \in \mathcal{V}$, some y_j-node is additionally labelled with the variable name x_i. Our intention is that y_j is additionally labelled with x_i in a model of α if v_i is coloured with colour j in a three-colouring induced by that model. We use ϕ to enforce that that two labels x_i, x_k are not placed on the node labelled with the same y_j if v_i and v_k are adjacent in $\mathcal{G}$, *i.e.*, they must have a different colour in the induced three colouring. Hence $\mathcal{G}$ can be three coloured if, and only if, α is satisfiable. Consequently, satisfiability is NP- and entailment coNP-hard in the extended assertion language

Finally, we briefly discuss allowing for existentially quantified variables in assertions. Formally, we amend the syntax of assertions to

$$\alpha ::= \exists x_1 \ldots x_n.(\phi, \sigma),$$

where $x_1, \ldots, x_n$ range over *Vars*. The semantics for an interpretation $\mathcal{I} = (V_b^{\mathcal{I}}, E_p^{\mathcal{I}}, \ell^{\mathcal{I}})$ is $\mathcal{I} \models \exists x_1 \ldots x_n.(\phi, \sigma)$ if, and only if, there exist $v_1, \ldots, v_n \in V_r^{\mathcal{I}}$ such that $\mathcal{I}' \models (\phi, \sigma)$, where $\mathcal{I}' = (V_r^{\mathcal{I}}, E_p^{\mathcal{I}}, \ell^{\mathcal{I}}[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n])$. It is easily seen that satisfiability in this extended fragment is still in polynomial time. However, it follows from recent results by Gorogiannis, Kanovich and O'Hearn on the complexity of abduction that entailment becomes coNP-hard [53].

6.4 Discussion

This chapter dealt with the computational complexity of entailment in a fragment of separation logic that was introduced in [11] and allows for reasoning about programs with linked lists. We improved the coNP algorithm given in [11] and showed that entailment is decidable in polynomial time. To this end, we showed that for any SL-formula we can compute in polynomial time a corresponding SL-graph in a particular normal form which has an equivalent corresponding SL-formula. Moreover, we showed that deciding entailment between two SL-formulae then reduces to checking for the existence of a homomorphism between their associated SL-graphs in normal form. A key advantage was that the homomorphism, if it exists, is uniquely determined by the SL-graphs, and that checking the homomorphism conditions can be performed in polynomial time. As a byproduct of the developed concepts, we obtained that satisfiability in the assertion language is in polynomial time. Finally, we discussed various natural syntactic extensions that lead to intractability of satisfiability or entailment.

As promised in the introduction, we close this chapter with discussing the differences between the syntax and semantics used in this thesis and in [11]. On a syntactic level, the difference between [11] and our assertion language is that [11] contains *nil* as an expression. This does however not give more expressiveness, since we can introduce a designated variable *nil* and implicitly join $nil \mapsto nil$ to every spatial assertion to obtain the same effect. On a semantic level, we have given the semantics of our assertion language in terms of SL-graphs, whereas it is given in terms of heaps and stacks in [11]. This is, however, only for technical convenience, since both semantic models are isomorph: red nodes of an interpretation can be viewed as the set of allocated heap cells, the set of p-edges of an interpretation as a representative of the contents of the heap cells and the variable labelling function as the stack. The main difference to [11] on a semantic level is that our semantics is *intuitionistic*. In [11], $\mathcal{I} \models \alpha = (\phi; \sigma)$ if, and only if, $\mathcal{I} \models \phi$ and $\mathcal{I} \models \sigma$, *i.e.*, the semantics is non-intuitionistic with respect to the definition provided by Reynolds [92]. Informally speaking, in our semantics models can contain more red nodes than actually required. It should, however, not be difficult to transfer the results obtained in this thesis to the semantics used in [11]. The concept of SL-graphs in normal form should adopt straight forwardly to non-intuitionistic semantics. However, the homomorphism conditions would require some adjustments. There basically needs to be an extra condition that ensures that when h is a homomorphism from G to G', all edges from G' are covered by h. These extra conditions would ensure that no model of $\alpha(G')$ can contain extra red nodes that, informally speaking, do not get used up by $\alpha(G)$. Furthermore, some adjustments would need to be made to cater for the precise semantics of lists used in [11], since our list semantics is imprecise. Working out the details is an interesting task for future work.

A further aspect of future work could be to identify syntactic fragments of extensions of our assertion language for which computing entailment remains in polynomial time. For example, regarding the extension of our assertion language with existential quantification, the hardness proof in [53] requires formulae that do not naturally occur in real-world program verification. Without going into too much detail, it seems conceivable that there exist fragments of this assertion language defined in terms of properties of SL-graphs for which entailment could be polynomial time computable.

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] Thomas Ball, Vladimir Levin, and Sriram K. Rajamani. A decade of software model checking with slam. *Communications of the ACM*, 54:68–76, 2011.

[5] Kshitij Bansal, Rémi Brochenin, and Etienne Lozes. Beyond shapes: Lists with ordered data. In Luca de Alfaro, editor, *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 425–439. Springer Berlin / Heidelberg, 2009.

[6] Yehoshua Bar-Hillel, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, (14):143–172, 1961.

[7] Matthias Beck. *The Arithmetic of Rational Polytopes*. PhD thesis, Temple University, 2000.

[8] A.P. Bel'tyukov. Decidability of the universal theory of natural numbers with addition and divisibility (in russian). *Zapiski Nauchnyh Seminarov LOMI*, 60:15–28, 1976.

[9] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a tool suite for automatic verification of real–time systems. In *Proceedings of the Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer Berlin / Heidelberg, 1995.

[10] Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter W. O'Hearn, Thomas Wies, and Hongseok Yang. Shape analysis for composite data structures. In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19th International Conference on Computer aided verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 178–192. Springer Berlin / Heidelberg, 2007.

[11] Josh Berdine, Cristiano Calcagno, and Peter OHearn. A decidable fragment of separation logic. In Kamal Lodaya and Meena Mahajan, editors, *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 110–117. Springer Berlin / Heidelberg, 2005.

[12] Josh Berdine, Cristiano Calcagno, and Peter OHearn. Smallfoot: Modular automatic assertion checking with separation logic. In Frank de Boer, Marcello Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Proceedings of the 4th International Symposium on Formal Methods for Components and Objects (FMCO'05)*, volume 4111 of *Lecture Notes in Computer Science*, pages 115–137. Springer Berlin / Heidelberg, 2006.

[13] Stanislav Böhm, Stefan Göller, and Petr Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In Paul Gastin and Franois Laroussinie, editors, *Proceedings of the 21st International Conference on Concurrency Theory*

(CONCUR'10), volume 6269 of *Lecture Notes in Computer Science*, pages 177–191. Springer Berlin / Heidelberg, 2010.

[14] Marius Boszga, Radu Iosif, Tomáš Vojnar, and Filip Konecny. FLATA. `http://www-verimag.imag.fr/FLATA.html`, 2011. Accessed January 11, 2012.

[15] Ahmed Bouajjani, Marius Bozga, Peter Habermehl, Radu Iosif, Pierre Moro, and Tomáš Vojnar. Programs with lists are counter automata. In Thomas Ball and Robert Jones, editors, *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 517–531. Springer Berlin / Heidelberg, 2006.

[16] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer Berlin / Heidelberg, 2008.

[17] Marius Bozga and Radu Iosif. On decidability within the arithmetic of addition and divisibility. In Vladimiro Sassone, editor, *Proccedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 425–439. Springer Berlin / Heidelberg, 2005.

[18] Marius Bozga, Radu Iosif, and Yassine Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2):275–303, 2009.

[19] Marius Bozga, Radu Iosif, and Swann Perarnau. Quantitative separation logic and programs with lists. *Journal of Automated Reasoning*, 45:131–156, 2010.

[20] Julian Bradfield and Colin Stirling. Modal mu-calculi. In Johan Van Benthem Patrick Blackburn and Frank Wolter, editors, *Handbook of Modal Logic*, vol-

ume 3 of *Studies in Logic and Practical Reasoning*, pages 721–756. Elsevier, 2007.

[21] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[22] Cristiano Calcagno, Dino Distefano, Peter OHearn, and Hongseok Yang. Space invading systems code. In Michael Hanus, editor, *Logic-Based Program Synthesis and Transformation (LOPSTR'08)*, volume 5438 of *Lecture Notes in Computer Science*, pages 1–3. Springer Berlin / Heidelberg, 2009.

[23] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In Ramesh Hariharan, V. Vinay, and Madhavan Mukund, editors, *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer Berlin / Heidelberg, 2001.

[24] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[25] Cristiana Chitic and Daniela Rosu. On validation of XML streams using finite state machines. In *Proceedings of the 7th International Workshop on the Web and Databases (WebDB'04)*, pages 85–90, New York, NY, USA, 2004. ACM.

[26] Andrew Chiu, George Davida, and Bruce Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications. Informatique Théorique et Applications*, 35(3):259–275, 2001.

[27] Alonzo Church. An unsolvable problem of elementary number theory. *The American Journal Of Mathematics*, 58:345–363, 1936.

[28] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella.

NuSMV 2: An opensource tool for symbolic model checking. In Ed Brinksma and Kim Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 241–268. Springer Berlin / Heidelberg, 2002.

[29] E. M. Clarke, M. Khaira, and X. Zhao. Word level model checking — avoiding the Pentium FDIV error. In *Proceedings of the 33rd annual Design Automation Conference (DAC'96)*, pages 645–648, New York, NY, USA, 1996. ACM.

[30] Edmund Clarke. The birth of model checking. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin / Heidelberg, 2008.

[31] Edmund Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer Berlin / Heidelberg, 1982.

[32] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In Kurt Jensen and Andreas Podelski, editors, *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer Berlin / Heidelberg, 2004.

[33] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2000.

[34] Hubert Comon and Yan Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In Alan Hu and Moshe Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer Berlin / Heidelberg, 1998.

[35] Byron Cook, Christoph Haase, Joël Ouaknine, Matthew Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In Joost-Pieter Katoen and Barbara König, editors, *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer Berlin / Heidelberg, 2011.

[36] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[37] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992.

[38] Zhe Dang, Oscar H. Ibarra, and Zhi-Wei Sun. On two-way nondeterministic finite automata with one reversal-bounded counter. *Theoretical Computer Science*, 330(1):59–79, 2005.

[39] Stéphane Demri and Régis Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, 19:1541–1575, 2009.

[40] Stéphane Demri, Ranko Lazic, and Arnaud Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theoretical Computer Science*, 411(22-24):2298–2316, 2010.

[41] Stéphane Demri and Arnaud Sangnier. When model-checking freeze LTL over counter machines becomes decidable. In C.-H. Luke Ong, editor, *Proceedings of the 13th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'10)*, volume 6014 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin / Heidelberg, 2010.

[42] The Coq development team. The Coq proof assistant reference manual, 2004. Accessed January 11, 2012.

[43] Dino Distefano and Matthew J. Parkinsonm. jstar: towards practical verification for java. In Gail E. Harris, editor, *Proceedings of the 23rd Conference on*

Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'08), pages 213–226, New York, NY, USA, 2008. ACM.

[44] Diego Figueira, Piotr Hofman, and Slawomir Lasota. Relating timed and register automata. In Sibylle B. Fröschle and Frank D. Valencia, editors, *Proceedings of the 17th International Workshop on Expressiveness in Concurrency (EXPRESS'10)*, volume 41 of *EPTCS*, pages 61–75, 2010.

[45] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[46] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik*, 37:349–360, 1930.

[47] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[48] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul Spirakis, editors, *Proceedings of the 37th international colloquium conference on automata, languages and programming (ICALP'10): Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer Berlin / Heidelberg, 2010.

[49] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Branching-time model checking of parametric one-counter automata. In Lars Birkedal, editor, *Proceedings of the 15th International Conference on Foundations of Software Science and Computer Security (FoSSaCS'12)*, volume 7213 of *Lecture Notes in Computer Science*, pages 406–420. Springer Berlin / Heidelberg, 2012.

[50] Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes. In Jean-Yves Marion and Thomas Schwentick, editors, *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 405–416, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[51] Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *Proceedings of the 24th Annual Symposium on Logic In Computer Science (LICS'09)*, pages 235–244. IEEE Computer Society Press, 2009.

[52] Georges Gonthier. A computer-checked proof of the four colour theorem. `http://research.microsoft.com/~gonthier/4colproof.pdf`, 2004. Accessed January 11, 2012.

[53] Nikos Gorogiannis, Max Kanovich, and Peter O'Hearn. The complexity of abduction for separated heap abstractions. In Eran Yahav, editor, *Proceedings of the 18th Internation Symposium on Static Analysis*, Lecture Notes in Computer Science, pages 25–42. Springer Berlin / Heidelberg, 2011.

[54] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer Berlin / Heidelberg, 2009.

[55] Christoph Haase and Carsten Lutz. Complexity of subsumption in the $\mathcal{EL}$ family of description logics: Acyclic and cyclic tboxes. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 25–29. IOS Press, 2008.

[56] Matthew Hague and Anthony Lin. Model checking recursive programs with numeric data types. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer Berlin / Heidelberg, 2011.

[57] Thomas Henzinger, Ranjit Jhala, Rupak Majumdar, and Grgoire Sutre. Software verification with blast. In Thomas Ball and Sriram Rajamani, editors, *Model Checking Software, Proceedings of the 10th International Spin Workshop (SPIN'03)*, volume 2648 of *Lecture Notes in Computer Science*, pages 235–239. Springer Berlin / Heidelberg, 2003.

[58] Ulrich Hertrampf, Clemens Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions (extended abstract). In *Proceedings of the 8th Annual Structure in Complexity Theory Conference*, pages 200–207, 1993.

[59] David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

[60] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

[61] Markus Holzer. On emptiness and counting for alternating finite automata. In *Developments in Language Theory (DLT'95)*, pages 88–97, 1995.

[62] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23:279–295, 1997.

[63] Oscar H. Ibarra and Zhe Dang. On two-way FA with monotonic counters and quadratic diophantine equations. *Theoretical Computer Science*, 312:359–378, January 2004.

[64] Oscar H. Ibarra and Zhe Dang. On the solvability of a class of diophantine equations and applications. *Theoretical Computer Science*, 352(1-3):342–346, 2006.

[65] Oscar H. Ibarra, Tao Jiang, Nicholas Q. Trân, and Hui Wang. New decidability results concerning two-way counter machines. *SIAM Journal on Computing*, 24(1):123–137, 1995.

[66] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of the 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 14–26, New York, NY, USA, 2001. ACM.

[67] Bart Jacobs and Frank Piessens. The VeriFast program verifier. Technical Report 520, Department of Computer Science, Katholieke Universiteit Leuven, 2008.

[68] Petr Jančar, Antonín Kučera, Faron Moller, and Zdeněk Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Information and Computation*, 188:1–19, 2004.

[69] Petr Jančar and Zdeněk Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Information Processing Letters*, 104(5):164 – 167, 2007.

[70] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: formal verification of an OS kernel. In *Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP'09)*, pages 207–220, New York, NY, USA, 2009. ACM.

[71] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *Research Report LSV-04-16*. LSV, ENS de Cachan, 2004.

[72] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In Philippa Gardner and Nobuko Yoshida, editors, *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer Berlin / Heidelberg, 2004.

[73] Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In Parosh Aziz Abdulla, Ahmed Bouajjani, and Markus Müller-Olm, editors, *Software Verification: Infinite-State Model Checking and Static Program Analysis*, number 06081 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[74] Leonhard Lipshitz. The diophantine problem for addition and divisibility. *Transactions of the American Mathematical Society*, 235:271–283, 1978.

[75] Leonhard Lipshitz. Some remarks on the diophantine problem for addition and divisibility. In *Proceedings of the Model Theory Meeting*, volume 33, pages 41–52, 1981.

[76] Richard Lipton. The reachability problem is exponential-space-hard. Technical report, Yale University, New Haven, CT, 1976.

[77] Yuri Matiyasevich. Enumerable sets are diophantine. *Doklady Akademii Nauk SSSR*, 191:279–282, 1970. English translation in Soviet Mathematics, Doklady, vol. 11, no. 2, 1970.

[78] Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the 13th Symposium on Theory of Computing (STOC'81)*, pages 238–246, New York, NY, USA, 1981. ACM.

[79] Gerome Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *Proceedings of the 21st Symposium on Principles of Database Systems (PODS'02)*, pages 65–76, New York, NY, USA, 2002. ACM.

[80] Marvin L. Minsky. Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. *The Annals of Mathematics*, 74(3):437–455, 1961.

[81] Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proceedings of the 32nd Conference on Programming Language Design and Implementation (PLDI'11)*, pages 556–566, New York, NY, USA, 2011. ACM.

[82] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[83] Russell O'Connor. Essential incompleteness of arithmetic verified by Coq. In Joe Hurd and Tom Melham, editors, *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics*, volume 3603 of *Lecture Notes in Computer Science*, pages 245–260. Springer Berlin / Heidelberg, 2005.

[84] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *Proceedings of the 10th Annual Conference on Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin / Heidelberg, 2001.

[85] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE'92)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer Berlin / Heidelberg, 1992.

[86] Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.

[87] Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[88] C. Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4), 2008.

[89] Wojciech Plandowski and Wojciech Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272, London, UK, 1999. Springer-Verlag.

[90] Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congres de Mathematiciens des Pays Slaves*, pages 92–101, 1929.

[91] J. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer Berlin / Heidelberg, 1982.

[92] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Anual Symposium on Logic in Computer Science (LICS'02)*, pages 55–74. IEEE Computer Society, Los Alamitos, CA, USA, 2002.

[93] Henry G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[94] Julia Robinson. Definability and decision problems in arithmetic. *Journal of Symbolic Logic*, 14(2):98–114, 1949.

[95] Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105 – 135, 1986.

[96] Barkley Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Mathematics*, 63(1):211–232, 1941.

[97] Philipe Schnoebelen. The complexity of temporal logic model checking. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Proceedings of the 4th Conference on Advances in Modal Logic*, pages 393–436, 2003.

[98] Olivier Serre. Parity games played on transition graphs of one-counter processes. In Luca Aceto and Anna Inglfsdttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 337–351. Springer Berlin / Heidelberg, 2006.

[99] Jeffrey Shallit. The Frobenius problem and its generalizations. In Masami Ito and Masafumi Toyama, editors, *Proceedings of the 12th International Conference on Developments in Language Theory (DLT'08)*, volume 5257 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin / Heidelberg, 2008.

[100] Michael Sipser. *Introduction to the theory of computation: second edition.* PWS Publishing, Boston, 2nd edition, 2006.

[101] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32:733–749, 1985.

[102] Craig Smorynski. *Logical number theory I: an introduction.* Springer-Verlag, 1991.

[103] Ales Smrčka and Tomáš Vojnar. Verifying parametrised hardware designs via counter automata. In Karen Yorav, editor, *Proceedings of the 3rd Haifa Veri-*

fication Conference, volume 4899 of *Lecture Notes in Computer Science*, pages 51–68. Springer Berlin / Heidelberg, 2008.

[104] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[105] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Annual Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.

[106] Heribert Vollmer. A generalized quantifier concept in computational complexity theory. In *Revised Lectures from the 9th European Summer School on Logic, Language, and Information: Generalized Quantifiers and Computation*, pages 99–123, London, UK, 2000. Springer-Verlag.

[107] Igor Walukiewicz. Model checking CTL properties of pushdown systems. In Sanjiv Kapoor and Sanjiva Prasad, editors, *Proceedings of the 20th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'00)*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer Berlin / Heidelberg, 2000.

[108] Gaoyan Xie, Zhe Dang, and Oscar Ibarra. A solvable class of quadratic diophantine equations with applications to verification of infinite-state systems. In Jos Baeten, Jan Lenstra, Joachim Parrow, and Gerhard Woeginger, editors, *Proceedings of the 30th International Colloquium om Automata, Languages and Programming (ICALP'03)*, volume 2719 of *Lecture Notes in Computer Science*, pages 668–680. Springer Berlin / Heidelberg, 2003.

[109] Jin yi Cai and Merrick L. Furst. PSPACE survives constant-width bottlenecks. *Internation Journal of Foundations of Computer Science*, 2(1):67–76, 1991.

Appendix A

Missing proofs

A.1 Missing proofs from Chapter 2

A.1.1 Proof of Serialisability of $\mathsf{EXPSPACE}$

The proof of $\mathsf{EXPSPACE}$-serialisability was established by Göller in an informal technical report accompanying [48]. As already stated in Chapter 2, we wish to repeat the proof of the theorem here in order to keep this thesis self-contained.

Recall that by Definition 4, given a complexity class C and a language $R \subseteq \{0,1\}^*$, a language $L \subseteq \Sigma^*$ is exponentially C-serialisable via R if there exists a polynomial p and a language $U \in \mathsf{C}$ such that for all $w \in \Sigma^n$ and $m = \exp p(n)$,

$$w \in L \Leftrightarrow \chi_U(w \cdot \mathrm{bin}_m(0)) \cdot \chi_U(w \cdot \mathrm{bin}_m(1)) \cdots \chi_U(w \cdot \mathrm{bin}_m(\exp\exp(p(n)) - 1)) \in R.$$

Theorem 2.4.2 states that for every $L \in \mathsf{EXPSPACE}$ there is a regular language R such that L is exponentially L-serialisable via R. The proof of this theorem builds upon results from [50], which are stated in terms of a polynomial version of serialisability given in the subsequent definition.

Definition 22 Given a complexity class C and a language $R \subseteq \{0,1\}^*$, a language $L \subseteq \Sigma^*$ is C-serialisable via R if there exists a polynomial p and a language $U \in \mathsf{C}$ such that for all $w \in \Sigma^n$ and $m = p(n)$,

$$w \in L \Leftrightarrow \chi_U(w \cdot \mathrm{bin}_m(0)) \cdot \chi_U(w \cdot \mathrm{bin}_m(1)) \cdots \chi_U(w \cdot \mathrm{bin}_m(\exp(p(n)) - 1)) \in R.$$

The following theorem follows as an immediate consequence from Theorem 22 in [50].

Theorem A.1.1 *For every $L \in \mathsf{PSPACE}$ there is some regular language R such that L is L-serialisable via R.*

Before we begin with the proof of Theorem 2.4.2, let us recall the following proposition which is folklore.

Proposition A.1.1 *For every language $L \subseteq \Sigma^*$ in $\mathsf{EXPSPACE}$ there is some polynomial q such that the padded language $L' \overset{def}{=} \{w \cdot \$^n : w \in L, n = \exp q(|w|)\} \subseteq \Sigma \dot\cup \{\$\}$ is in PSPACE.*

We are now going to prove Theorem 2.4.2. Let $L \subseteq \Sigma^*$ be a language in $\mathsf{EXPSPACE}$ and assume without loss of generality that $\{0, 1, \$\} \cap \Sigma = \emptyset$. By Proposition A.1.1, there exists a polynomial q and a language $L' = \{w \cdot \$^n : w \in L, n = \exp q(|w|)\}$ such that $L' \in \mathsf{PSPACE}$. Theorem A.1.1 yields a polynomial p', an $R' \in \mathsf{REG}$ and $U' \in \mathsf{L}$ such that for each $w' \in (\Sigma \dot\cup \{\$\})^n$ and $m' = p'(n)$ we have

$$w' \in L' \Leftrightarrow \chi_{U'}(w \cdot \mathrm{bin}_{m'}(0)) \cdot \chi_{U'}(w \cdot \mathrm{bin}_{m'}(1)) \cdots \chi_{U'}(w \cdot \mathrm{bin}_{m'}(\exp(p'(n)) - 1)) \in R.$$

Theorem 2.4.2 requires us to provide $U \in \mathsf{L}$, $R \in \mathsf{REG}$ and a polynomial p. Choose p such that for all $n \in \mathbb{N}$, $\exp p(n) > p'(n + \exp(q(n))) + 1$. The language U consists of all words $u \in (\Sigma \dot\cup \{0, 1\})^*$ such that u can factored as

$$u = w \cdot b \cdot z \cdot 0^j, \tag{$\star$}$$

where $j \in \mathbb{N}$, $w \in \Sigma^n$, $b \in \{0, 1\}$, $z \in \{0, 1\}^{p'(n+\exp(q(n)))}$ such that $b = 1$ implies $w \cdot \$^{\exp(q(n))} \cdot z \in U'$, *i.e.*, $b = 0$ or $w \cdot \$^{\exp(q(n))} \cdot z \in U'$. Let us argue that $U \in \mathsf{L}$. First, checking whether u is of the form ($\star$) clearly can be performed in logarithmic space. Second, p grows sufficiently large in order to simulate any logarithmically-space bounded Turing machine $\mathcal{M}$ that decides L' in L due to our choice of p.

It remains to provide the regular language R, which we are construct from R'. Suppose that

$$b_1 b_2 \cdots b_n = \chi_{U'}(w \cdot \mathrm{bin}_{m'}(0)) \cdot \chi_{U'}(w \cdot \mathrm{bin}_{m'}(1)) \cdots \chi_{U'}(w \cdot \mathrm{bin}_{m'}(\exp(p'(n)) - 1)).$$

By construction of U, for $m = \exp p(n)$ we have that

$$\chi_U(w \cdot \mathrm{bin}_m(0)) \cdot \chi_U(w \cdot \mathrm{bin}_m(1)) \cdots \chi_U(w \cdot \mathrm{bin}_m(\exp\exp(p(n)) - 1)) = \\ 1b_1 1b_2 \cdots 1b_n 00 \cdots 00.$$

Thus, when reading $1b_1 1b_2 \cdots 1b_n 00 \cdots 00$ in *pairs*, *i.e.*, as $(1b_1)(1b_2) \cdots (1b_n)(00) \cdots (00)$, whenever the first component is 1 we read information relevant for the simulation of R'. Recall that regular languages are closed under homomorphisms. We set $R \overset{\text{def}}{=} h(R' \cdot \{\Box\}^*)$, where $h : \{0, 1, \Box\} \to \{0, 1\}$ is a homomorphism such that $h(0) = 10$, $h(1) = 11$ and $h(\Box) = 00$. This finishes the proof of Theorem 2.4.2.