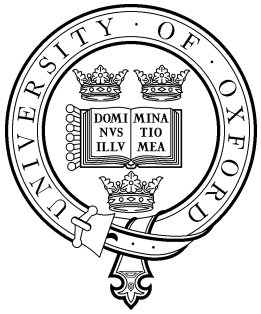
Algorithmic Verification Problems in Automata-Theoretic Settings



Daniel Bundala

Hertford College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity Term 2014

To my parents, for always supporting me.

—

To my sister, for all her encouragement.

Acknowledgements

First of all, let me thank my supervisor Joël Ouaknine. I was really lucky to have won the supervisors' lottery; this acknowledgement is too short to contain all the support, guidance and the amount of freedom Joël has given me.

I am further indebted to James Worrell for many discussions as well as for serving as my Transfer and Confirmation examiner. I thank Stefan Göller for agreeing to serve as my external examiner and Luke Ong for being my internal as well as Transfer and Confirmation examiner.

I thank Engineering and Physical Sciences Research Council (EPSRC) for funding my DPhil.

Finishing of this thesis would not be possible without the support and encouragement from my family and the many friends I have met in Oxford. Especially, let me thank Jakub Závodný—the short run through the University Parks on 14 May 2010 has had profoundly influenced me ever since. František Simančík—for being a great source of inspiration, wisdom and ridiculous postcards. Rastislav Lenhardt—for showing me the importance of "ťah na bránku" in ice hockey as well as in life. Ventsi Chonev who helped tremendously in proofreading of this thesis. And many other great Czech & Slovak as well as rowing, football, ice hockey, ultimate frisbee, cycling and bouldering friends I was lucky enough to meet while pursuing my DPhil.

Finally, let me thank my parents for their continuous support throughout all my years in Oxford and my sister Saša for all her encouragement.

Abstract

Problems in formal verification are often stated in terms of finite automata and extensions thereof. In this thesis we investigate several such algorithmic problems.

In the first part of the thesis we develop a theory of completeness thresholds in Bounded Model Checking. A completeness threshold for a given model M and a specification φ is a bound k such that, if no counterexample to φ of length k or less can be found in M, then M in fact satisfies φ. We settle a problem of Kroening *e*t al. [KOS$^+$11] in the affirmative, by showing that the linearity problem for both regular and ω-regular specifications (provided as finite automata and Büchi automata respectively) is PSPACE-complete. Moreover, we establish the following dichotomies: for regular specifications, completeness thresholds are either linear or exponential, whereas for ω-regular specifications, completeness thresholds are either linear or at least quadratic in the recurrence diameter of the model under consideration.

Given a formula in a temporal logic such as LTL or MTL, a fundamental problem underpinning automata-based model checking is the complexity of evaluating the formula on a given finite word. For LTL, the complexity of this task was recently shown to be in NC [KF09]. In the second part of the thesis we present an NC algorithm for MTL, a quantitative (or metric) extension of LTL, and give an AC^1 algorithm for UTL, the unary fragment of LTL. We then establish a connection between LTL path checking and planar circuits which, among others, implies that the complexity of LTL path checking depends on the Boolean connectives allowed: adding Boolean exclusive or yields a temporal logic with P-complete path-checking problem.

In the third part of the thesis we study the decidability of the reachability problem for parametric timed automata. The problem was introduced over 20 years ago by Alur, Henzinger, and Vardi [AHV93]. It is known that for three or more parametric clocks the problem is undecidable. We translate the problem to reachability questions in certain extensions of parametric one-counter machines. By further reducing to satisfiability in Presburger arithmetic with divisibility, we obtain decidability results for several classes of parametric one-counter machines. As a corollary, we show that, in the case of a single parametric clock (with arbitrarily many nonparametric clocks) the reachability problem is NEXP-complete, improving the nonelementary decision procedure of Alur *et al.*. The case of two parametric clocks is open. Here, we show that the reachability is decidable in this case of automata with a single parameter.

Contents

Chapter 1

Introduction

> *At a recent real-time computer-programming conference, the participants were given a question to answer:*
>
> *— "If you had just boarded an airliner and discovered that your team of programmers had been responsible for the flight control software, how many of you would disembark immediately?"*
>
> *Among the forest of raised hands only one man sat motionless. When asked what he would do, he replied that he would be quite content to stay aboard. With his team's software, he said, the plane was unlikely to even taxi as far as the runway, let alone take off.*

Unlike in the joke, a computer bug may have real consequences. The European Space Agency's Ariane V famously exploded a handful of seconds after the first launch due to a bug in the system operating the rocket. It was estimated that 17% of web servers certified by certificate authorities were vulnerable at some point to a serious security bug that could be exploited to obtain passwords and private keys of users. The bug, later named *Heartbleed*, was caused by insufficient input test leading to a buffer over read.

Planes, rockets and helicopters have crashed, passwords have been revealed and incorrect floating-point divisions have been caused by bugs in computer systems. One of the main goals of formal verification is to guarantee that such and similar bugs do not occur: given a computer system can we guarantee that the system satisfies the given specification? For example, "Does the airplane take off?", "Does the division algorithm meet the IEEE-754 standard?", "Is every request eventually followed by an acknowledgement?".

These and similar questions lie at the heart of formal verification. Over the last two

decades, the field of formal verification has shown a lot of progress and success. For example, both the bug responsible for Ariane V explosion and the Pentium division error were detectable by technology available at the time.

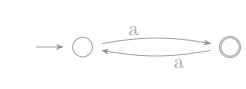
In formal verification we try to guarantee that software and computer systems behave correctly as the functioning of more and more devices depends on the correctness of software. From a theoretical point of view, Turing [Tur36] and Rice [Ric53] showed that there is no general algorithm that, given a computer program, can determine whether the program terminates, let alone meets a more complex specification (expressible in a suitable way). In practice, however, the situation is more positive and often the particular verification problem, once abstracted appropriatelly, reduces to a decidable problem.

Given a computer system C, one can view C as a transition system T with the states of T corresponding to possible configurations of C and transitions between states encoding possible changes of configurations as prescribed by C. Such transition systems are often modelled as various automata depending on the features we model.

The simplest case is when T is a finite automaton, modelling a finite state system. If the system contains a stack or counters, the system can be more suitably modelled using pushdown or counter automata [Min67]. Markov chains are a popular formalism for specifying stochastic systems. Extending finite automata with clocks yields timed automata [AD94] that allow one to express timing aspects of the system. Further, hybrid automata [Hen96] have been proposed to model systems combining discrete and continuous parts, with almost all combinations of these various classes of automata having been studied in the literature [KNSS02, BFS09, BBE$^+$10, Spr00].

On the other hand, a specification is a collection of executions of the system that are deemed to be good. Verification problems are then often phrased as checking whether executions of the model are in the specification. Two popular ways of expressing specifications have emerged. First, the specification is expressed by an automaton, the language of which equals the specification. Depending on the property being expressed, different types of automata can be used.

Second, temporal logics (LTL, MTL, TPTL, $\mathsf{FO}[<]$, PCTL, etc.) have been devised that are capable of expressing properties about individual executions and relationships between individual points of the execution. In the latter case, the specification is the collection of all possible executions satisfying the given formula. A natural and robust approach for dealing with properties expressed as formulae φ in a temporal logic is to build an automaton B_φ, whose language is the set of the executions satisfying φ.



In both cases, a problem originating in formal verification reduces to a problem involving automata. In fact, many verification problems reduce to problems about automata. For example, to check whether a system modelled as an automaton M meets a specification φ, it suffices to check whether

$$L(M) \subseteq L(B_\varphi),$$

where B_φ is the automaton recognising executions satisfying φ. We study this and related problems in this thesis.

The breadth of problems in formal verification is immense. These range from engineering problems, such as how to implement and represent in memory various automata, to algorithmic and more foundational problems, such as determining the complexity or even the decidability of various problems. In this thesis we focus on problems of the latter types.

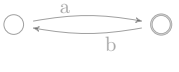
1.1 Scope and Contributions of This Thesis

We now describe various automata-based problems arising in formal verification that are studied in this thesis.

1.1.1 Completeness Thresholds in Bounded Model Checking

One of the most commonly used bug-finding techniques in formal verification is bounded model checking (BMC), first introduced in [BCCZ99, BCC+03]. Within three or four years following its introduction, BMC was found to have almost entirely replaced BDD-based model checking in the hardware industry, thanks largely to the huge advances made in SAT technology over the past 10 to 15 years.

Given a model M and a specification φ, in order to check whether $L(M) \subseteq L(B_\varphi)$, BMC considers only paths up to some predetermined length k. Such an approach serves to find a counterexample: a word of length at most k in $L(M) \setminus L(B_\varphi)$. However, the absence of a counterexample is inconclusive; a genuine bug could still lurk deeper within the system. For this reason, from the very inception of the technique, researchers have attempted to turn BMC into a *complete* method with the ability also to guarantee the absence of counterexamples of any length. See, for instance, the original work of Biere *et al.* [BCCZ99], or the 2008 Turing Award lecture of Ed Clarke [CES08], in which the problem is described as a topic of active research.



A *completeness threshold* for a given model M and specification φ is a bound k such that, if no counterexample to φ of length k or less can be found in M, then M in fact satisfies φ. Formally, it is an integer k such that if $L_k(M) \subseteq L_k(B_\varphi)$ then $L(M) \subseteq L(B_\varphi)$ where $L_k(X)$ is a restriction of $L(X)$ to words of lengths at most k.

In [BCCZ99], Biere *et al.* observed that for simple safety properties of the form Gp, a *completeness threshold* is given by the *diameter* (longest distance between any two states) of M: indeed, if no counterexample to Gp of length at most the diameter of the system can be found, then no counterexample of any length can possibly exist. Likewise, for liveness properties such as Fq, the *recurrence diameter* (length of the longest loop-free path) of M can be seen to be an adequate completeness threshold.

In a recent paper [KOS$^+$11], Kroening *et al.* substantially extend these observations by identifying a large class of ω-regular specifications for which completeness thresholds *linear* in the recurrence diameter of the models can be effectively computed. This class consists of so-called *cliquey* Büchi automata, and subsumes among others the fragment of LTL consisting of unary next-free formulas. The authors also present examples of simple specifications having quadratic or even exponential completeness thresholds.[1]

However, [KOS$^+$11] left as an open question the decidability of determining whether an ω-regular specification has a linear completeness threshold.

We study this problem in Chapter 3. The completeness threshold is best stated in terms of automata. A completeness threshold k for a given model M and a specification φ is an upper bound on the shortest accepting path in the automaton $M \times B_{\neg\varphi}$.

We say that an automaton[2] B has *linear completeness threshold* if there exists $c \in \mathbb{R}^+$ such that for all models M, the length of the shortest accepting path (sap) is bounded by a multiple of $\mathrm{rd}(M)$: $\mathrm{sap}(M \times B) \leq c\,\mathrm{rd}(M)$. Therefore, to show that B is not linear, we have to find an infinite family of models M_i such that $\mathrm{sap}(M_i \times B) \geq i\,\mathrm{rd}(M)$. On the face of it, it is not clear that the existence of such an infinite family of models is even decidable. However, we show that nonlinearity is always witnessed by a special class of models. The class, moreover, possesses a small-model property and we show that there are arbitrarily large M_i's if and only if there exists an M_i for a sufficiently large (but bounded) i.

This characterisation of automata with linear completeness thresholds allows us to establish the following results:

[1] The precise definition of the magnitude of completeness thresholds is given in Chapter 3.

[2] Automaton B recognises the complement of the property: $L(B) = L(B_{\neg\varphi})$.

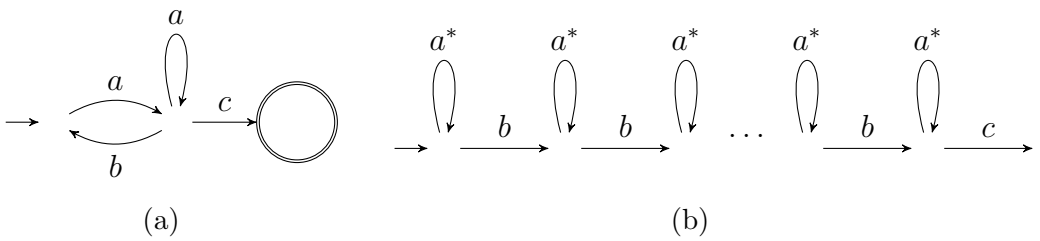


Figure 1.1: A finite automaton with nonlinear completeness threshold (a) and family of models witnessing nonlinearity as constructed in Chapter 3 (b).

(i) Showing whether the completeness threshold of an automaton is linear is decidable and, in fact, PSPACE-complete. The result applies both to finite and ω-regular specifications (provided as automata and Büchi automata respectively).

(ii) For regular specifications, completeness thresholds are either linear or exponential. Moreover, we show that exponential completeness thresholds are asymptotically the worst possible.

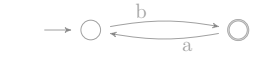
(iii) For ω-regular specifications, completeness thresholds are either linear or at least quadratic (and can be exactly quadratic and exponential).

1.1.2 Path Checking of Temporal Logics

Given a property φ specified in a temporal logic (such as LTL) and a system M, a goal of formal verification is to check the inclusion $L(M) \subseteq L(B_\varphi)$. To accommodate different requirements and various extensions of M, the problem has been extensively studied for various types of automata modelling the system. Finite, Büchi, timed, one-counter, pushdown, probabilistic,... and almost any combination thereof have been considered. The complexity of these model checking problems is typically at least PSPACE-hard.

In Chapter 4 we take the opposite approach and instead of extending the automata and making them more powerful, we restrict the models in the hope of obtaining faster model checking algorithms.

One of the simplest, yet still meaningful, cases of automata to consider are the so-called path automata. A *path automaton* M is a linear sequence of states with a transition from each state to its successor. If M is deterministic then such an automaton corresponds to precisely one word w and thus checking the inclusion $L(M) \subseteq L(B_\varphi)$ reduces to checking whether the word w satisfies the property φ. This is the so-called *path-checking* problem, sometimes known at the membership problem: $w \models \varphi$?



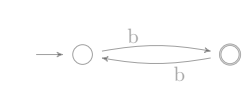
Not only is the problem of evaluating a formula on a given word one of the fundamental problems for the logic the formula is expressed in, but this problem also plays a key role in the design and analysis of offline monitoring and runtime verification procedures [FS04, MN04]. The path-checking problem also appears in testing [ABG+05] and in Monte-Carlo-based probabilistic verification [YS02].

Although the path-checking problem is simply stated, determining its precise complexity can prove to be quite challenging. The case of LTL was investigated more than a decade ago [DS02, MS03], and at the time is was conjectured that the straightforward polynomial-time dynamic-programming algorithm is not optimal.[3] And indeed, using reductions to planar circuits and tree-contraction algorithms, it was recently proved [KF09] that LTL path checking allows an efficient parallel algorithm and lies in NC—in fact, in $\mathsf{AC}^1[\mathsf{logDCFL}] \subseteq \mathsf{AC}^2$. (This seminal result was awarded the ICALP 2009 best-paper award.)

We study the path-checking problems for various temporal logics in Chapter 4. The LTL path-checking algorithm [KF09] uses the fact that temporal operators can be represented by planar monotone upward stratified circuits (Figure 1.2) and then combines the circuits using a tree contraction algorithm on the parse tree of φ. A general algorithm for evaluating circuits of this type is then invoked. It was conjectured in [KF09] that by devising a specialised algorithm tailored to the specific circuits emerging in the construction it may be possible to improve the upper bound even further. We show this is unlikely as the evaluation of a large class of planar circuits is reducible to LTL path checking. This reduction sheds further light on the importance of the *monotonicity* of all Boolean and temporal LTL operators, as the evaluation of planar non-monotone circuits is known to be P-complete and hence unlikely to be in the AC hierarchy. We use this to show that adding a single non-monotone Boolean operator into LTL makes path checking P-time complete. This is also the simplest and most natural extension of LTL we are aware of whose path-checking problem is higher than that for pure LTL

Recently, the work [KF09] on LTL path checking was extended [KF12] to a very restricted metric extension of LTL, in which only temporal operators of the form $\mathrm{U}_{\leq b}$ are allowed. In this thesis, we extend the construction in [KF09] and give an algorithm for full Metric Temporal Logic (MTL) with the same complexity—$\mathsf{AC}^1[\mathsf{logDCFL}]$—as for LTL.

[3] The best known lower bound for LTL path checking is NC^1, which crudely arises from the NC^1-hardness of mere Boolean formula evaluation.



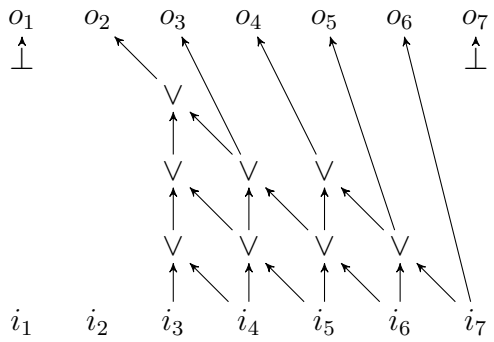


Figure 1.2: Example of a planar monotone upward stratified circuit constructed in Chapter 4.

An examination of the algorithmic constructions of [KF09] further shows that the most intricate parts arise in handling the Until operator. We show that the removal of binary operators from the logic, yielding Unary Temporal Logic (UTL), leads to a much simpler path-checking problem, enabling us to devise an AC^1 algorithm for UTL path checking.

To summarise, at the time of writing, the results in Chapter 4 provide[4]:

(i) the most expressive known extension of LTL with an NC path-checking algorithm, namely MTL.

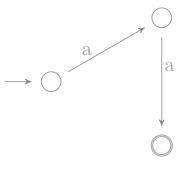
(ii) the simplest known extension of LTL with a strictly harder path-checking problem, namely LTL + Xor, and

(iii) the most expressive known fragment of LTL with a strictly more efficient path-checking algorithm than for full LTL, namely UTL.

1.1.3 Halting Problem for Parametric Timed Automata

Embedded computer systems often depend on the precise timing of individual events. A variety of such systems have been modelled using timed automata—a class of finite automata extended with clocks. All clocks of a timed automaton evolve simultaneously and at the same rate, and each clock can be reset individually by every edge of the automaton. Edges of timed automata are further labelled by expressions over the clocks thereby limiting the clock values for which the particular edge can be taken.

[4]Subject to standard complexity-theoretic assumptions.



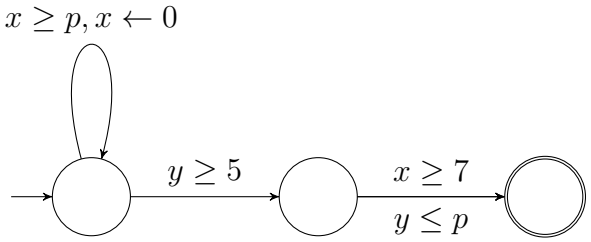


Figure 1.3: An example of a parametric timed automaton with two clocks $\{x, y\}$ and a single parameter p. The final state is reachable if, for example, $p = 10$.

Thus expressions on the edges constrain the possible evolutions of systems modelled as timed automata. Getting such timing constraints right can however be difficult. Therefore, a natural problem to consider is to leave some of the constraints open, specify them by parameters, and then try to find values of parameters for which the timed automaton behaves as required.

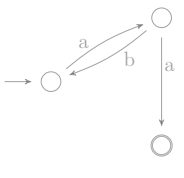
Similarly, timed automata might operate in a surrounding environment, so leaving some constraints open allows us to compute for which values of the environment the system behaves as required.

Introducing parameters brings another level of complexity to standard verification problems. Already the decidability of the simplest parametric problem—parametric reachability—for timed automata has been open for more than twenty years [AHV93].

The problem of reachability in parametric timed automata was introduced over two decades ago in a seminal paper of Alur, Henzinger, and Vardi [AHV93]: given a timed automaton in which some of the constants appearing within guards on transitions are parameters, is there some assignment of integers to the parameters such that an accepting location of the resulting concrete timed automaton becomes reachable? Such a reachability decision procedure can, for example, be used to verify any safety property of the parametric timed automaton: is a bad state reachable for some values of parameters?

In this framework, a clock is said to be *nonparametric* if it is never compared with a parameter, and is *parametric* otherwise. Alur *et al.* [AHV93] showed that, for timed automata with a single parametric clock, reachability is decidable (irrespective of the number of nonparametric clocks). The decision procedure given in [AHV93] however has provably nonelementary complexity. In addition, [AHV93] showed that reachability becomes undecidable for timed automata with at least three parametric clocks.

The decidability of reachability for parametric timed automata with *two* parametric clocks (and arbitrarily many nonparametric clocks) was left open in [AHV93],



with hardly any progress (partial or otherwise) that we are aware of in the intervening period. Alur *et al.* [AHV93] showed that this problem subsumes the question of reachability in Ibarra *et al.*'s "simple programs" [IJTW93], also open for over 20 years, as well as the decision problem for a fragment of Presburger arithmetic with divisibility.

In this thesis we tackle the problem of one and two parametric clocks. We show in Chapter 5 that the problems reduce to the reachability problem for particular extensions of parametric one-counter machines [HKOW09]. We achieve this by restricting our attention to parametric timed automata with *closed* (i.e., *non-strict*) clock constraints. As parameters are restricted to range over the integers,[5] standard digitisation techniques apply [HMP92], reducing the reachability problem over dense time to discrete (integer) time. (Alternately, our results also apply directly to timed automata interpreted over discrete time, regardless of the type of constraints used.) The restriction to integer time enables us, among others, to keep track of the values of two parametric clocks using a single counter, in effect reducing the reachability problem for timed automata with two parametric clocks to a halting problem for parametric one-counter machines.

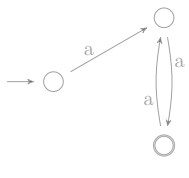
The counter-machine formulation is more convenient to analyse. Then in Chapter 6, building upon new developments in the theory of one-counter machines [HKOW09] and their encodings in Presburger arithmetic [Haa12], we show decidability of the reachability problem in certain classes of the resulting parametric one-counter machines.

In more detail, we show in Chapter 5 that:

(i) The reachability problem for parametric timed automata with a single parametric clock is equivalent to the reachability problem for parametric one-counter machines with updates $-1, 0$ and 1, and comparisons $\leq p_i, \geq p_i$ for parameters p_i.

(ii) The reachability problem for parametric timed automata with two parametric clocks is equivalent to the reachability problem for parametric one-counter machines with updates $\pm c_i, \pm p_i$, comparisons $\leq p_i, \geq p_i$ and a few other technical operations for constants $c_i \in \mathbb{N}$ and parameters p_i.

[5] Other researchers have considered variations in which parameters are allowed to range over rationals, yielding different outcomes as regards the decidability of reachability; see, e.g., [Mil00, Doy07], discussed further below.



Then in Chapter 6 we use the relationship between parametric timed automata and the respective classes of parametric one-counter machines to show that:

(iii) In the case of a single parametric clock (with arbitrarily many nonparametric clocks and arbitrarily many parameters), we show that the reachability problem is NEXP-complete, improving the nonelementary decision procedure of Alur *et al.*

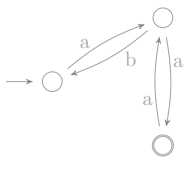
(iv) The reachability problem is decidable for the class of parametric one-counter machines mentioned in (i).

(v) The reachability problem is decidable for parametric timed automata with two parametric clocks (and arbitrarily many nonparametric clocks), if the automaton uses only a *single* parameter. Further, the problem is $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hard.

(vi) We show that the reachability problem is decidable for the class of parametric one-counter machines mentioned in (ii) if they use only a single parameter.

1.2 Publications

Some of the results presented in this thesis have already appeared as peer-reviewed papers in the proceedings of computer-science conferences. Specifically, the work on completeness thresholds in bounded model checking (Chapter 3) appeared in a LICS 2012 paper [BOW12] coauthored with my supervisor Joël Ouaknine and with James Worrell. Results on the path checking in temporal logics (Chapter 4) appear in an ICALP 2014 paper [BO14b] coauthored with my supervisor. Finally, results on the relationship between parametric timed automata and parametric-one counter machines (Chapter 5) as well as decidability results for parametric timed automata with one parametric clock (Section 6.1) appear in an MFCS 2014 paper [BO14a] also coauthored with my supervisor.



Chapter 2

Preliminaries

2.1 Automata Theory

In this thesis we study automata-based problems arising in formal verification. We now define the different types of automata studied in this thesis.

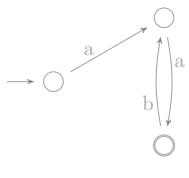
In general, an automaton is a finite collection of *states* and *edges* (*transitions*) from one state to another. Formally,

Definition 2.1.1. *An* automaton A *is tuple* $A = (S, s_0, \Delta, F, Op, \lambda)$ *where*

- S *is the set of states,*
- s_0 *is the initial state,*
- $\Delta \subseteq S \times S$ *is the set of edges,*
- F *is the set of final states and*
- $\lambda : \Delta \to Op$ *is a function assigning a label from set* Op *to every edge.*

Different types of automata studied in this thesis differ only in the labelling function λ and the semantics of the transitions. In particular, whenever we define a class of automata in this thesis, we specify only the set Op of allowed labels and their respective semantics.

For example, a finite automaton describes a language of finite words over a given alphabet. Formally, an *alphabet* Σ is a finite set of symbols. A *word* w of length $n \in \mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ over the alphabet Σ is a sequence of n symbols $w(1)w(2)w(3)\ldots$ from the alphabet, i.e., $w(i) \in \Sigma$. The *empty word* is denoted by ϵ and the length of w is denoted by $|w|$. Then:



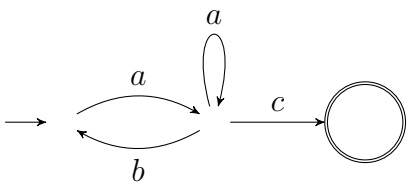


Figure 2.1: A finite automaton.

Definition 2.1.2. A (nondeterministic) finite automaton[1] *A over the alphabet Σ is an automaton with*

$$Op = \Sigma.$$

An example of a finite automaton is shown in Figure 2.1. If A is an automaton, then we use A to denote both the automaton and (assuming the context is clear) its set of states. In particular, we denote by $|A|$ the number of states of the automaton A.

2.1.1 Working with Paths

In this thesis we study semantic properties of automata. Such properties are tightly linked to paths through the automata and we now introduce notation useful for describing paths and their properties.

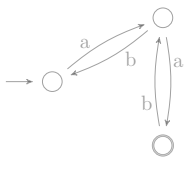
A *transition* $e = (q, q')$ of A is a tuple in the transition relation Δ where $q \in Q$ is the initial state of the transition, $\lambda(e) \in \Sigma$ is the letter associated with the transition and $q' \in Q$ is the final state of the transition. Two transitions (q_1, q_2) and (q_3, q_4) are *consecutive* if the final state of the first transition equals the initial state of the second transition: $q_2 = q_3$.

A *path* π through an automaton A is a sequence of transitions $(e_i)_{i=1}^{i=l}$ such that the transitions e_i and e_{i+1} are consecutive. The length of π, denoted $|\pi|$, equals l. The word $\lambda(e_1), \lambda(e_2), \ldots, \lambda(a_l)$ spelled by (associated with) π is denoted by $\text{word}(\pi)$. A path is called *finite* if $|\pi|$ is a natural number $|\pi| \in \mathbb{N}$ and infinite otherwise.

A path through finite automaton A is called *accepting* if it starts in the initial state q_0 and finishes in a final states $s \in F$. A word w is called accepting if there is an accepting path π through A such that the word associated with π equals w, i.e., $\text{word}(\pi) = w$. The *language* of A, denoted $L(B)$, is the set of all accepting words of A: $L(A) = \{w \in \Sigma^* \mid w \text{ is accepting}\}$.

We now present the operations on paths used throughout the thesis. If π is nonempty (length at least 1) path then we use $\text{first}(\pi) = q_1$ to denote the *initial state*

[1]For simplicity and the ease of presentation, the automata considered in this thesis do not have ϵ edges



of π. We say that π starts (begins) in q_1. If π is finite and nonempty then we use $\text{last}(\pi) = q'_l$ to denote the *final state* of π. We say that π finishes (terminates) in q'_l. If π is finite, we often write $\pi : q_1 \to q'_l$. If s and t are states of A and $w \in \Sigma^*$ is a word we write $s \xrightarrow{w} t$ if there is a path $\pi : s \to t$ such that $\text{word}(\pi) = w$.

If the path π is written as $\pi = (e_i)_{i=1}^{i=l}$ and $\rho = (f_i)_{i=1}^{i=k}$ is a path such that the transitions e_l and f_1 are consecutive then the *concatenation* $\pi \cdot \rho$ of π and ρ is obtained by appending ρ after π. That is $\pi.\rho = (g_i)_{i=1}^{i=l+k}$ where $g_i = e_i$ if $i \leq l$ and $g_i = f_{i-l}$ if $i > l$.

If $i \in \mathbb{N}$ is an index then $\pi(i)$ denotes the i-th vertex (starting from 1) of π. Precisely, $\pi(1) = q_1$ and $\pi(i) = q'_{i-1}$ for $i > 1$. For $a \leq b \in \mathbb{N}$ the expression $\pi[a \dots b]$ denotes the subpath of π from index a to index b inclusively. The expression $\pi[a \dots)$ denotes the suffix of π starting at index a.

A path π of length at least one is a *loop* if the first and the last state coincide: $\text{first}(\pi) = \text{last}(\pi)$ and $|\pi| > 0$. If π is a loop and $k \in \mathbb{N}$ then by π^k we denote the path obtained by concatenating π with itself k times: $\pi^k = \underbrace{\pi.\pi \dots \pi}_{k \text{ times}}$. The path π^ω is the infinite path obtained by concatenating π with itself ω-many times. Formally, $\pi^\omega = (g_i)_{i=1}^{i=\infty}$ where $g_i = e_{((i-1) \bmod l)+1}$. A path is *simple* (loop-free) if it visits every state at most once.

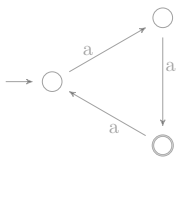
An infinite path π is lasso-shaped if it can be written as $\pi = \pi_1 \pi_2^\omega$. A finite path π is k-*bounded* if $|\pi| \leq k$. An infinite lasso-shaped path π is k-*bounded* if it can be written as $\pi = u.v^\omega$, where $|u| + |v| \leq k$.

2.2 Different Types of Automata[†]

2.2.1 Timed Automata

Runs of finite automata are linear sequences of symbols from the alphabet (events). Thus, finite automata define only relative order of events but completely abstract the times of when the individual events occur. Suppose, for example, that we want to model as an automaton a controller in a car. Then not only we may require in the model that brakes are applied after the brake pedal is pressed, but we may want to specify that the brakes are applied within some small time delay after pressing the pedal. The behaviour of this and many other systems often depends on the timing of individuals events. *Timed automata* [AD94] have emerged as a popular formalism for modelling formalism such system.

[†]All automata models introduced in this thesis are also summarised in Appendix A.



A timed automaton is a finite automaton extended with a finite set of clocks C that all progress at the same rate and that can be individually reset to zero. Moreover, every transition is labelled by an expression of the form $\bigwedge_i c \bowtie d$ where $c \in C$ is a clock, $\bowtie \in \{\leq, =, \geq\}$ and $d \in \mathbb{N}$ is a constant. The set of all such expression over the set C of clocks is denoted $G(C)$.

Definition 2.2.1. *A* timed automaton A *over the set of clocks* C *is an automaton with*

$$Op = 2^C \times G(C).$$

For each edge $e = (s, s')$ with $\lambda(e) = (R, G)$, the state s is the initial state, state s' is the final state, set R specifies which clocks are reset by taking the edge and G is the guard specifying for which clock values the edge is enabled.

The configuration of a timed automaton depends on the state the automaton is in as well as the current values of clocks. Formally, a *configuration* (s, ν) of A consists of a state s and function $\nu : C \to T$ assigning a value from the time domain T to each clock. Timed automata can be interpreted either over dense time ($T = \mathbb{R}_{\geq 0}$) or over discrete time ($T = \mathbb{N}_{\geq 0}$).

The initial clock valuation ν_0 assigns 0 to every clock and the initial configuration is (s_0, ν_0).

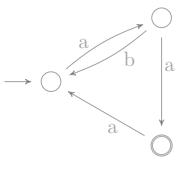
An execution of a timed automaton is a combination of staying in the same state letting the time evolve and (instantaneous) state transition by taking an edge. Precisely, a transition exists from configuration (s, ν) to (s', ν') in A ,written $(s, \nu) \to (s', \nu')$, if either

- there exists $t \in T$ such that $\nu(c) + t = \nu'(c)$ for every clock $c \in C$
- or there is an edge $e = (s, s') \in E$ with $\lambda(e) = (R, G)$ such that G is satisfied for current clock values ν and $\nu'(c) = \begin{cases} 0 & \text{if } c \in R \\ \nu(c) & \text{if } c \notin R \end{cases}$

The former transition corresponds to elapse of time t whereas the latter corresponds to taking the edge e.

A *run* of a timed automaton is a sequence $\pi = \boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_k}$ of configurations such that $\boldsymbol{c_i} \to \boldsymbol{c_{i+1}}$ for each i. A run is called *accepting* if $\boldsymbol{c_1}$ is the initial configuration and $\boldsymbol{c_k}$ is in a final state.

As described in the introduction, requiring all guards in a timed automaton to be concretely specified is too restrictive. We now introduce parametric extension of timed automata which allows the constraints to be parametrised.



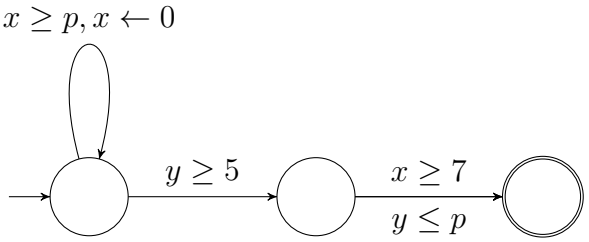


Figure 2.2: A parametric timed automaton with two clocks $\{x, y\}$ and a single parameter p. The final state is reachable if, for example, $p = 10$.

Formally, let P be a finite set of *parameters*. Then for the set of clocks C, the expression $G(C, P)$ denotes the set of guards (similarly to $G(C)$) where the clocks can be compared to constants as well as to parameters: $\bigwedge_i c \bowtie d$ where $c \in C$ is a clock, $\bowtie \in \{\leq, =, \geq\}$ and $d \in \mathbb{N} \cup P$.

Definition 2.2.2. *A* parametric timed automaton A *over the set of clocks* C *and parameters* P *is an automaton with*

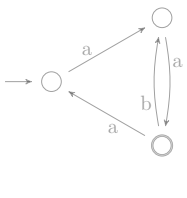
$$Op = 2^C \times G(C, P).$$

An example of a parametric timed automaton is shown in Figure 2.2. By instantiating parameters, we obtain timed automata.

An *assignment* for P is a function $\gamma : P \to \mathbb{N}$ assigning a *natural number* to each parameter. Given a parametric timed automaton A and an assignment γ, the expression A^γ denotes the timed automaton obtained by instantiating every parameter $p \in P$ at $\gamma(p)$. One of the main problems studied in this thesis is the problem of finding a parameter valuation such that a given parametric timed automaton has an accepting run, i.e., for some input values of parameters, is there a terminating computation for the parametric timed automaton? Formally,

Problem 2.2.3.
Name: Existential Halting Problem
Input: *Parametric Timed Automaton* A
Problem: *Is there an assignment* γ *such that* A^γ *has an accepting run?*

The existential halting problem is also known in literature [AHV93] as the parametric reachability or the emptiness problem. We omit "existential" in this thesis and write simply "halting problem". We say that two automata A_1 and A_2 have *equivalent halting problem* if A_1 halts if and only if A_2 halts.



2.2.2 One-Counter Machines

Counter machines are obtained from finite automata by adding counters. We deliberately use the word "machine" to describe automata extended with counters as this disambiguates presentation in Chapters 5 and 6 where we show a relationship between timed automata and counter machines and study the properties of the former using the latter.

Each counter of a counter machine ranges over natural numbers and each edge can increment, decrement or test for zero each of the counters independently. It is well-known [Min67] that having two counters makes the machine as computationally powerful as a Turing machine.

In this dissertation, we focus on problems related to reachability in (parametric) timed automata. We will prove that such problems are equivalent to reachability in certain classes of (parametric) counter machines with only a single counter.

Definition 2.2.4. *A* one-counter machine C *is an automaton with*

$$Op = \{+c, -c, =c, \geq c \ : \ c \in \mathbb{N}\}.$$

Unlike two-counter machine, the reachability problem for one-counter machines is decidable and in fact in NP [HKOW09] even when the constants are encoded in binary. A *configuration* (s, x) of C consists of a state $s \in S$ and counter value $x \in \mathbb{N}$. That is, the counter is always required to stay nonnegative. The initial configuration of C is $(s_0, 0)$. A configuration (s, x) is accepting if $s \in F$ is final state.

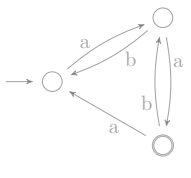
A configuration (s', x') is reachable in one step from (s, x) (written $(s, x) \to (s', x')$) in C if there exists an edge $e = (s, s') \in E$ such that

- if $\lambda(e) = \pm c$ then $x \pm c = x'$
- if $\lambda(e) = \sim c$ then $x = x'$ and $x \sim c$ where $\sim \in \{=, \geq\}$

A $r$un of a one-counter automaton is a sequence $\pi = \boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_k}$ of configurations such that $\boldsymbol{c_i} \to \boldsymbol{c_{i+1}}$ for each index i. A run is called $a$ccepting if $\boldsymbol{c_1}$ is the initial configuration and $\boldsymbol{c_k}$ is in a final state.

Notice that the '$\geq c$' edge is a syntactic sugar for a gadget consisting of a '$-c$' edge followed by a '$+c$' edge. To show equivalence between parametric timed automata and parametric one-counter machine we shall also require the other comparison '$\leq c$' edge. This is analogous to nonparametric settings where an equivalence was established previously [HOW12].

An *simple bounded one-counter machine* M is obtained by extending the codomain Op of the function λ assigning an operation to each edge by '$\leq c$' operation for $c \in \mathbb{N}$.



Definition 2.2.5. *A* simple bounded one-counter machine C *is an automaton with*

$$Op = \{+c, -c, =c, \geq c, \leq c \ : \ c \in \mathbb{N}\}.$$

Configuration (s', x') is reachable in one step from configuration (s, x) using a '$\leq c$' transition

- if $\lambda(e) = \leq c$ then $x = x'$ and $x \leq c$

It was shown in [FJ13] that allowing '$\leq c$' edges makes the reachability problem PSPACE-complete.

In order to show the reductions from parametric timed automata we also need two more types of operations: '$\equiv 0 \mod c$' and '$+[0, c]$' for $c \in \mathbb{N}$. Configuration (s', x') is reachable in one step from configuration (s, x) using these edges provided it satisfies:

- if $\lambda(e) = +[0, c]$ then $x \leq x' \leq x + c$
- if $\lambda(e) = \equiv 0 \mod c$ then $x = x'$ and $x \equiv 0 \mod c$

Definition 2.2.6. *A* bounded one-counter machine[2] C *is an automaton with*

$$Op = \{+c, -c, =c, \geq c, \leq c, +[0, c], \equiv 0 \mod c \ : \ c \in \mathbb{N}\}.$$

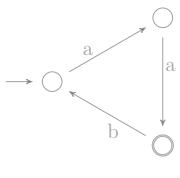
We use $\text{counter}(s, x) = x$ to denote the counter value in the configuration (s, x). We extend the definition to runs componentwise: if π is a run in C then $\text{counter}(\pi)$ is a sequence of natural numbers obtained by applying counter to each element of π. We write $\text{counter}(\pi) \leq C$ (resp. $\text{counter}(\pi) \geq C$) if the comparison holds for every element: $\forall i \ . \ \text{counter}(\pi(i)) \leq C$ (resp. $\forall i \ . \ \text{counter}(\pi(i)) \geq C$).

Each run in C modifies the counter. By $\text{effect}(\pi)$ we denote the change in the counter: $\text{effect}(\pi) = \text{counter}(\text{last}(\pi)) - \text{counter}(\text{start}(\pi))$. A run π is called *positive* if $\text{effect}(\pi) > 0$. It is called *negative* if $\text{effect}(\pi) < 0$. Note that if π is a loop and $a \in \mathbb{N}$ then $\text{effect}(\pi^a) = a \, \text{effect}(\pi)$.

Further, for a run π we define $\text{div}(\pi) = \max_i |\, \text{counter}(\pi(i)) - \text{counter}(\text{first}(\pi))|$ to be the maximum difference between the counter and its initial value in the run π. If a run π can be written as $\pi = \pi_1 \pi_2$ then it follows from the triangle inequality that $\text{div}(\pi) \leq \text{div}(\pi_1) + \text{div}(\pi_2)$.

A *path* in C is a path in the underlying graph. Given an initial configuration (s, x), a path π starting in a state s uniquely determines a run in C. The run follows

[2]Note that our definition is more general than the one given in [HOW12].



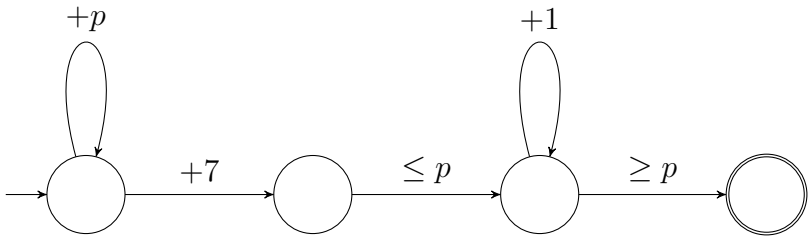


Figure 2.3: Example of a parametric bounded one-counter machine. The final state is reachable for all $p \in [7, \infty)$.

edges as given by π and the counter value is updated accordingly. Thus, if the initial configuration is known, we often identity runs with paths and vice versa.

To reason about parametric timed automata, we introduce parametric one-counter machines. Formally, let P be a finite set of *parameters*. An *assignment* for P is a function $\gamma : P \to \mathbb{N}$ assigning a *natural number* to each parameter. Then

Definition 2.2.7. *A* parametric bounded one-counter machine C *over the set of parameters* P *is an automaton with*

$$Op = \{+c, -c, +p, -p, \leq c, = c, \geq c, \leq p, = p, \geq p, +[0, p], \equiv 0 \bmod c \ : \ c \in \mathbb{N}, p \in P\}.$$

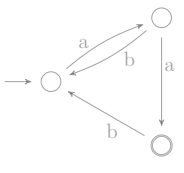
We deliberately omitted the parametric version of '$\equiv 0 \bmod p$' as such an operation is not needed in the reduction from parametric timed automata to parametric bounded one-counter machines. Parametric version of simple one-counter machines is obtained in the analogous way.

Definition 2.2.8. *A* simple parametric bounded one-counter machine C *over the set of parameters* P *is an automaton with*

$$Op = \{+c, -c, +p, -p, \leq c, = c, \geq c, \leq p, = p, \geq p \ : \ c \in \mathbb{N}, p \in P\}.$$

Given a parametric one-counter machine C and an assignment γ the expression C^γ denotes the one-counter machine obtained by instantiating every parameter $p \in P$ at $\gamma(p)$. As for timed automaton, we have an equivalent notion of the halting problem for counter machines.

Problem 2.2.9.
Name: Existential Halting Problem
Input: *Parametric Bounded One-Counter Machine* C
Problem: *Is there an assignment* γ *such that* C^γ *has an accepting run?*



2.2.3 Büchi Automata

Finite automata recognise languages of finite words. We now describe Büchi automata which are the counterpart of finite automata recognising languages of infinite words. The structure of Büchi automata is the same as that of finite automata. Formally,

Definition 2.2.10. A Büchi automaton *over alphabet* Σ *is an automaton with*

$$Op = \Sigma.$$

The property differentiating Büchi automata is the acceptance condition. A Büchi automaton accepts only infinite paths. An infinite path π is accepted if:

- π starts in the initial state $\text{first}(\pi) = q_0$,
- Some final state $f \in F$ is visited infinitely often by π. Formally, we have: $\forall i \,.\, \exists j > i \,.\, \pi(j) = f$.

Büchi automaton accepts an infinite word $w \in \Sigma^\omega$ if there is an infinite accepting path π such that $\text{word}(\pi) = w$. The language of B is, analogously to finite automata, the set of all words $\{w \in \Sigma^\omega \mid w \text{ is accepted by } B\}$ accepted by B.

The language of B has the following property.

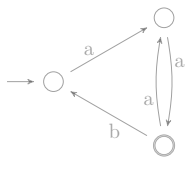
Lemma 2.2.11. *Suppose that* $L(B) \neq \emptyset$*. Then there is a lasso-shaped accepting path (and hence a lasso-shaped accepting word) in* B*.*

Proof. Since the language is nonempty, there is a word w in the language of B. Hence, there is an accepting path π in B. By definition, there exists some final state f of B that is visited infinitely often by π. In particular, the state f is visited twice and so there are indices $i < j$ such that $\pi(i) = \pi(j) = f$.

Notice that the subpath $\pi[i \dots j]$ is a loop in B that visits a final state. Hence, $\pi[i \dots j]^\omega$ is a valid path that visits a final state infinitely often. By construction, the sequence $\pi[1 \dots i](\pi[i \dots j]^\omega)$ is a valid lasso-shaped path visiting a final state infinitely often. □

2.3 Automata-Theoretic Verification

As mentioned in the introduction, one of the main goals of formal verification is to determine whether a given model M satisfies a given specification φ. *Specification*, is a collection C of allowed executions of the system specified in a suitable way.



Hence, to determine whether the system satisfies the property C, we wish to determine whether every behaviour of M is contained in C. That is, we wish to check $L(M) \subseteq C$. A lot of research in formal verification deals with how to perform this check most efficiently for various types of systems and formalisms expressing the specification.

A computer system can be viewed, generalising the different types of automata defined above, as a transition system between (possibly infinitely many) configurations of the computer system. Formally,

Definition 2.3.1. *A* transition system M *over* Σ *is a tuple* (S, s_0, Δ) *where*

- S *is the set of states,*
- s_0 *is the initial state,*
- $\Delta \subseteq Q \times \Sigma \times Q$ *is the transition relation.*

For example, configurations of one-counter machines (timed automata) correspond to states of (infinite-state) transition systems. Moreover, the relation $\boldsymbol{c_1} \to \boldsymbol{c_2}$ between configurations defines the transition relation.

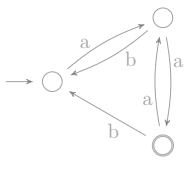
A transition system M captures the evolution of a computer system and thus we define the language $L(M)$ of the system to be the set of all possible finite (infinite) words generated by paths starting in the initial state s_0 of M. That is, $L(M) = \{\text{word}(\pi) \mid \exists \pi \in M \;.\; \text{first}(\pi) = s_0\}$.

Specifications are commonly given by an automaton A the language $L(A)$ of which equals the set of allowed executions . Thus, the problem of checking whether the model satisfies the specification reduces to checking whether $L(M) \subseteq L(A)$. Equivalently, if B is the automaton recognising the complement of A, that is $L(B) = \Sigma^\omega - Ł(A)$ then the problem reduces to checking whether $L(M) \cap L(B) = \emptyset$.

Now, the standard product construction that builds an automaton accepting the above intersection exists. Formally, the *product* of a transition system $M = (Q, s_0, \Delta)$ with an automaton $B = (Q', I', \Delta', F)$, denoted $M \times B$, is an automaton over Σ defined to be $(Q'', I'', \Delta'', F'')$ where the two transition systems synchronise on edges.

- $Q'' = Q \times Q'$
- $I'' = \{s_0\} \times I'$
- $e'' = ((q_1, q_1'), (q_2, q_2')) \in \Delta''$ and $\lambda(e'') = a$

  provided: there is $e = (q_1, q_2) \in \Delta$ with $\lambda(e) = a$ and
  there is $e' = (q_1', q_2') \in \Delta$ with $\lambda'(e) = a$,



- $F'' = Q \times F$.

It is well known that $L(M \times B) = L(M) \cap L(B)$. Thus, the problem of checking $L(M) \cap L(B) = \emptyset$ reduces to checking the existence of a word in the language of the product automaton, which in turn reduces to finding a path from the initial to a final state or a lasso-shaped paths visiting a final state infinitely often. Both problems can be easily solved in nondeterministic logarithmic space.

Observe that a word $w \in L(M) \cap L(B)$ is an execution of M that violates the specification. In such cases, we say that w is a counterexample (bug) to A in M.

2.4 Temporal Logics

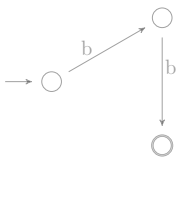
A popular way of describing specifications is via temporal logics, which have been specifically devised to express properties between points of individual execution. The most prominent logic is the Linear Temporal Logic (LTL) used for specifying ω-regular properties. We first define a more general logic (Metric Temporal Logic) and then derive LTL as its sublogic.

Let AP be a set of atomic propositions, $p \in \text{AP}$ and $I \subseteq \mathbb{R}_{\geq 0}$ be an interval with endpoints in $\mathbb{N} \cup \{\infty\}$. The formulae of *Metric Temporal Logic* (MTL) are defined recursively over the set of atomic propositions using Boolean connectives and temporal operators X (Next), Y (Previous), U (Until), S (Since), R (Release) and T (Trigger).

$$\varphi = p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathrm{X}_I\, \varphi \mid \mathrm{Y}_I\, \varphi \mid \varphi\, \mathrm{U}_I\, \varphi \mid \varphi\, \mathrm{S}_I\, \varphi \mid \varphi\, \mathrm{R}_I\, \varphi \mid \varphi\, \mathrm{T}_I\, \varphi$$

Formulae of MTL are evaluated over traces. We now define how an MTL formula φ can be evaluated over trace of (possibly infinite) length n. In Chapter 4 we deal with the problem of evaluating a formula on a given finite trace.

Formally, a *trace* π over AP of length $n \in \mathbb{N} \cup \{\infty\}$ is a function $\pi : \{1, \ldots, n\} \times \text{AP} \to \mathbb{B}$ assigning a truth value to every $p \in \text{AP}$ at every index. To evaluate MTL formulae on π, we further associate with π a sequence of strictly-increasing *timestamps* $t_1 < \ldots < t_n$. Given an MTL formula φ and index $1 \leq i \leq n$, the satisfaction relation



$\pi, i \models \varphi$ is defined recursively as follows.

$$
\begin{array}{ll}
\pi, i \models p & \text{if } p(i) = \top \\
\pi, i \models \neg\varphi & \text{if it is not the case that } \pi, i \models \varphi \\
\pi, i \models \varphi_1 \wedge \varphi_2 & \text{if } \pi, i \models \varphi_1 \text{ and } \pi, i \models \varphi_2 \\
\pi, i \models \varphi_1 \vee \varphi_2 & \text{if } \pi, i \models \varphi_1 \text{ or } \pi, i \models \varphi_2 \\
\pi, i \models X_I\varphi & \text{if } i+1 < n \text{ and } t_{i+1} - t_i \in I \text{ and } \pi, i+1 \models \varphi \\
\pi, i \models Y_I\varphi & \text{if } i > 1 \text{ and } t_i - t_{i-1} \in I \text{ and } \pi, i-1 \models \varphi \\
\pi, i \models \varphi_1 U_I \varphi_2 & \text{if } \exists j \,.\, (i \leq j \leq n) \wedge \left(\begin{array}{l} \pi, j \models \varphi_2 \\ t_j - t_i \in I \\ \forall k \,.\, i \leq k < j \implies \pi, k \models \varphi_1 \end{array} \right) \\
\pi, i \models \varphi_1 S_I \varphi_2 & \text{if } \exists j \,.\, (i \geq j \geq 1) \wedge \left(\begin{array}{l} \pi, j \models \varphi_2 \\ t_i - t_j \in I \\ \forall k \,.\, i \geq k > j \implies \pi, k \models \varphi_1 \end{array} \right)
\end{array}
$$

Other operators are expressible using the following semantic equalities: $\varphi \,\mathrm{R}_I\, \psi = \neg(\neg\varphi \,\mathrm{U}_I\, \neg\psi)$ and $\varphi \,\mathrm{T}_I\, \psi = \neg(\neg\varphi \,\mathrm{S}_I\, \neg\psi)$.

We say that a trace π *satisfies* (models) formula φ if $\pi, 1 \models \varphi$. The language associated with the given MTL formula is the set of all traces π such that $\pi, 1 \models \varphi$.

We now introduce two unary temporal operators: F (Eventually) and G (Globally), the semantics of these operators is defined via the following equalities: $\mathrm{F}_I\varphi = \top \,\mathrm{U}_I\, \varphi$, $\mathrm{G}_I\varphi = \neg\, \mathrm{F}_I\, \neg\varphi$. That it $\pi, i \models \mathrm{F}_I\varphi$ if φ holds at some point in the interval $t_i + I$ and $\pi, i \models \mathrm{G}_I\varphi$ if φ holds at all point in the interval $t_i + I$.

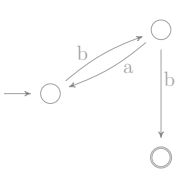
Linear Temporal Logic (LTL) is the subset of MTL in which I is always $[0, \infty)$ and is therefore omitted from the syntax. Moreover, the values of timestamps are irrelevant and hence ignored.The fragment UTL of LTL consists of all Boolean connectives and unary $(\mathrm{X}, \mathrm{F}, \mathrm{G})$ temporal operators and their past duals.

A problem studied in Chapter 4 is the problem of evaluating a given formula on a given trace.

Definition 2.4.1. *The* path-checking problem *for logic $\mathcal{L}$ is to determine, given a finite trace π and a formula φ of $\mathcal{L}$, whether $\pi, 1 \models \varphi$.*

It is well known [VW94, GPVW95] that given an LTL formula φ there is a Büchi automaton B recognising φ. That is, $L(B) = L(\varphi)$. In the worst case, the automaton is exponential in size in the length of φ.

The case of MTL is more subtle. In general, there are MTL formulae (e.g., $\mathrm{G}(a \implies F_{[1,1]}b)$) the language of which is not recognised by a timed automaton. Moreover, formulae of MTL are closed under negation whereas the languages of timed automata are not closed under complement [AD94]. On the other, if all intervals I in φ are



not punctual (contain more than one point) then it is known [AFH96] that φ is recognisable by a timed automaton.

LTL temporal operators satisfy the following identities:

$$\begin{aligned}
\varphi \,\mathrm{U}\, \psi &= \psi \vee (\varphi \wedge \mathrm{X}(\varphi \,\mathrm{U}\, \psi)) \\
\mathrm{F}\varphi &= \varphi \vee \mathrm{X}\,\mathrm{F}\,\varphi \\
\mathrm{G}\varphi &= \varphi \wedge \mathrm{X}\,\mathrm{G}\,\varphi \\
\varphi \,\mathrm{U}\, \psi &= \neg\big[(\neg\varphi)\,\mathrm{R}\,(\neg\psi)\big] \\
\varphi \,\mathrm{S}\, \psi &= \neg\big[(\neg\varphi) T (\neg\psi)\big]
\end{aligned}$$

Note that we defined MTL formulae in such a way that the negation can be applied only to atomic propositions. However, the last two identities above show that this is not a limitation as negation can always be pushed inwards. A formula in which negation is applied only to atomic propositions is said to be in the *negation normal form*. It follows that every formula can be translated into an equivalent one in negation normal form.

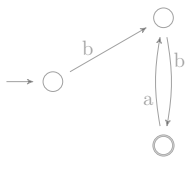
A *formula context* $\varphi(X)$ is a formula with one occurrence of a proposition replaced by a variable X. If $\psi(X)$ is another formula context then $(\varphi \circ \psi)(X)$ is the context obtained by substituting $\psi(X)$ for X in $\varphi(X)$. If $q \in \mathrm{AP}$ is a proposition then $\varphi(q)$ is obtained by substituting q for X. For example, $((p \,\mathrm{U}\, X) \circ (X \,\mathrm{S}\, q))(r) = (p \,\mathrm{U}\, (X \,\mathrm{S}\, q))(r) = p \,\mathrm{U}\, (r \,\mathrm{S}\, q)$. Observe that, composing formula contexts increases the size linearly as a formula context contain only one occurrence of X.

2.5 Circuits

The results in Chapter 4 depend on the theory of Boolean circuits. We now introduce basic notions; for a thorough introduction into the theory of Boolean circuits see e.g., [AB09].

A *Boolean circuit* (C, δ) consists of a set of *gates* C and a *predecessor* function $\delta : C \to \mathcal{P}(C)$ assigning to every gate g a collection of gates the gate g depends on (see Figure 2.4). If $d \in \delta(c)$ then we say c *depends* on d or that there is a *wire* from d to c. The number of gates depending on a gate c is called the *fan-in* of c.

With each circuit C one can naturally associate a directed graph. The vertices of the graph are the gates of C and there is an edge from vertex g to vertex h if $h \in \delta(g)$.



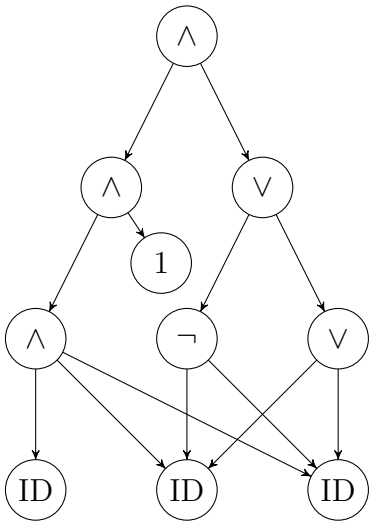


Figure 2.4: An example of a Boolean circuit.

For the circuit (C, δ) to be valid it is required that the graph is acyclic. The depth of a circuit is the length of a longest path in the graph.

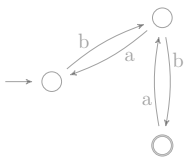
Further, a *type* is associated with each gate. The type can be OR, AND, NOT, ID, ONE or ZERO. The type determines how the output of a gate depends on the outputs of its predecessors. If c is of type τ and $\delta(c) = \{c_1, \ldots, c_n\}$ then we write $c = (\tau, c_1, \ldots, c_n)$.

The ONE and ZERO gates provide constants inputs. A gate is an *input* gate if it does not have a predecessor. We require that ONE and ZERO are the only input gates. The values of the remaining gates are defined recursively. Let v be a function assigning a value to each gate. Then v is defined as follows. Let g be a gate of type τ. Then

$$v(g) = \begin{cases} \bigwedge_{h \in \delta(g)} v(h) & \text{if } \tau = \text{AND} \\ \bigvee_{h \in \delta(g)} v(h) & \text{if } \tau = \text{OR} \\ \neg v(\delta(g)) & \text{if } \tau = \text{NOT} \\ v(\delta(g)) & \text{if } \tau = \text{ID} \\ \top & \text{if } \tau = \text{ONE} \\ \bot & \text{if } \tau = \text{ZERO} \end{cases}$$

A gate is an *output* gate if it is not a predecessor of any other gate. Given a circuit with one output gate, the *circuit value problem*, abbreviated as *CVP*, is the problem of determining the value of the output gate.

A circuit is *monotone* if it has no NOT gates. It is *planar* if the underlying DAG is planar. In this thesis, all edges (wires) are straight-line segment and so a *planar embedding* is induced by a function $\gamma : C \to \mathbb{R}^2$ assigning a point in the plane to



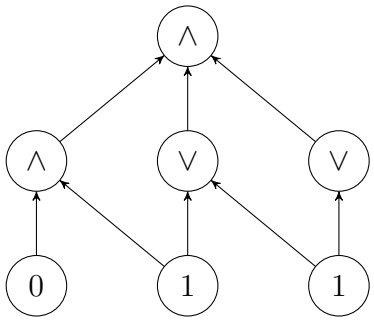


Figure 2.5: An example of a Boolean circuit.

every gate. For example, the drawing in Figure 2.5 is planar whereas the drawing in Figure 2.4 is not.

2.6 Complexity Classes

Throughout this dissertation, we make use of many complexity classes. See for example [Sip96] for an introduction into the basic complexity theory. The standard complexity classes used in the thesis are:

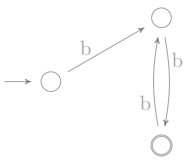
L	deterministic	logarithmic	space
NL	nondeterministic	logarithmic	space
P	deterministic	polynomial	time
NP	nondeterministic	polynomial	time
PSPACE	deterministic	polynomial	space
NPSPACE	nondeterministic	polynomial	space
EXP	deterministic	exponential	time
NEXP	nondeterministic	exponential	time

It is well known that

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

We also make use of several less standard complexity classes. It is easy to see that a log-space Turing Machine T has at most polynomially many different configurations. Thus, if T terminates, it terminates within at most polynomially many steps[3]. However, suppose that T is equipped with an additional stack, which does not count towards the logarithmic space limit. The stack can become unbounded and indeed there are instances of such Turing machines that require superpolynomially many steps to terminate.

[3]This shows that $\mathsf{L} \subseteq \mathsf{P}$.



Now, the class $\mathsf{logDCFL}$ consists of problems decidable by deterministic logspace Turing machines equipped with a stack such that the Turing machine always terminates in polynomial time.

The addition of the stack enables the Turing machine to recognise deterministic context-free languages. In fact [Sud78], the class $\mathsf{logDCFL}$ consists precisely of problems that are logspace many-one reducible to deterministic context-free languages (hence the name).

On the other hand, the class logCFL consists of problems that are logspace many-one reducible to (arbitrary) context-free languages.

For two complexity classes $\mathcal{C}$ and $\mathcal{D}$, the notation $\mathcal{C}^{\mathcal{D}}$ denotes the *oracle complexity class* of problems solvable by a Turing machine running in the complexity class $\mathcal{C}$ and making oracle calls to problems in the complexity class $\mathcal{D}$. We refer the reader to [AB09] for a detailed introduction to oracle complexity classes. For example, it is known that $\mathsf{PSPACE}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$ [AKRR11].

For the results in Chapter 4 we have to introduce complexity classes for Boolean circuits. Note that a single circuit has a fixed number of inputs. Thus, an n input-gate circuit can decide only inputs of length at most n. A language in general, however, contains words of arbitrary length. This motivates the introduction of families of circuits—with one circuit for each number of inputs. Let $\mathcal{C} = \{C_0, C_1, C_2, C_3, \ldots\}$ be a family of circuits with C_k having precisely k inputs. Then a (binary) language L is accepted by $\mathcal{C}$ if and only of

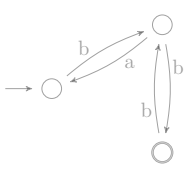
$$w \in L \iff C_{|w|}(w) = \top.$$

However, such a definition is too powerful. Consider the language

$$L_u = \{\{0,1\}^k \mid k\text{-th Turing machine halts on empty string}\}.$$

Now, L_u is undecidable [Sip96], on the other hand, there is a family of circuits accepting L_u. Namely C_k is either constant 1 or constant 0 depending on whether the k-th Turing machine halts or not. Thus, one considers only *uniform* families of circuits. A family of circuits $\mathcal{C} = \{C_0, C_1, C_2, \ldots\}$ is uniform if there is a log-space Turing machine T such that given input of length k, the machine T outputs the circuit C_k. For a more detailed introduction into Boolean circuits and circuit complexity classes see [AB09].

The most standard circuit complexity classes are AC^i and NC^i. The circuit class AC^i for $i \in \mathbb{N}$ consists of problems accepted by uniform and polynomial-size circuits



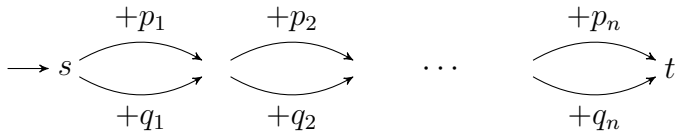


Figure 2.6: A parametric one-counter automaton.

of depth $\log^i$. The circuit class NC^i is a restriction of AC^i where it is required that the fan-in of every gate is at most 2.

Recall that the definition of circuits allows only very few types of gates. Now, a gate takes a set of Boolean inputs and produces an output which is a Boolean function of the inputs. In general, one can allow more complicated functions f than the Boolean conjunction, disjunction or negation. The computational power of the resulting circuits then depends on the complexity class C the function f can be evaluated in.

Given a language S and a complexity class C, we write $S \in \mathsf{AC}^1[C]$ if there is a family of AC^1 circuits with additional C-oracle gates that accept S. It is known that

$$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \begin{matrix}\mathsf{NL} \\ \mathsf{logDCFL}\end{matrix} \subseteq \mathsf{AC}^1 \subseteq \mathsf{AC}^1[\mathsf{logDCFL}] \subseteq \mathsf{AC}^2 \subseteq \cdots \subseteq \mathsf{AC}^i \subseteq \mathsf{AC}^{i+1} \subseteq \cdots \subseteq \mathsf{P}$$

2.7 Presburger Arithmetic

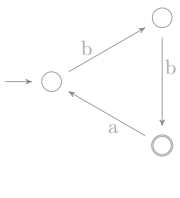
Presburger Arithmetic with Divisibility is the first-order logical theory over the structure $\langle \mathbb{N}, <, +, |, 0, 1 \rangle$. The existential fragment (formulae of the form $\exists x_1, x_2, \ldots, x_k.\varphi$ where φ has no quantifiers) is denoted as $\exists$PAD.

For example, the relation '$x \equiv y \bmod z$' can be expressed by a $\exists$PAD formula:

$$\varphi(x, y, z) := \exists p \,.\, (z|p \ \wedge \ x - p = y \ \wedge \ y < z).$$

The satisfiability of $\exists$PAD formulae was shown decidable in [Lip78, Bel80] and in NP [Lip81].

In [HKOW09, Haa12] it was shown how that the reachability relation in parametric one-counter machines is $\exists$PAD definable. Consider the parametric one-counter automaton C in Figure 2.6 and suppose we want to write a $\exists$PAD formula expressing the existence of a run in C from the configuration $(s, 0)$ to the configuration $(t, 0)$. For each of exponentially many path from s to t, we can write a simple $\exists$PAD formula



expressing the existence of a run along that path. E.g., the path only taking $+p_i$ edges corresponds to the formula:

$$\varphi_1 := \big[p_1 + p_2 + \ldots + p_n = 0 \ \wedge \ p_1 \geq 0 \ \wedge \ p_1 + p_2 \geq 0 \ \wedge \ \ldots \ \wedge \ p_1 + p_2 + \ldots p_{n-1} \geq 0\big].$$

Similarly, the path that takes $+q_1$ but then only takes only $+p_i$'s edges afterward correspond to the formula:

$$\varphi_2 := \big[q_1 + p_2 + \ldots + p_n = 0 \ \wedge \ q_1 \geq 0 \ \wedge \ q_1 + p_2 \geq 0 \ \wedge \ \ldots \ \wedge \ q_1 + p_2 + \ldots p_{n-1} \geq 0\big].$$

Therefore, the reachability in C from $(s, 0)$ to $(t, 0)$ can be expressed by the $\exists$PAD formula:

$$\varphi(p_1, \ldots, p_n, q_1, \ldots, q_n) := \bigvee_{i=1\ldots 2^n} \varphi_i.$$

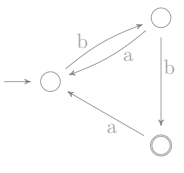
Now, φ is exponentially large in n and thus applying the generic NP satisfiability algorithm on φ would result in a nondeterministic algorithm running in exponential time in n. However, there is a natural NP algorithm for checking whether there is a run from $(s, 0)$ to $(t, 0)$: guess a formula φ_i and then check its satisfiability. These considerations motivate the introduction of $\exists$PAD *definable* sets.

Given a set $S \subseteq \mathbb{N}^k$ we say that S is $\exists$PAD *definable* if there is a finite set R of $\exists$PAD formulae[4], each formula with free variables $y_1, \ldots y_k$ such that $(n_1, \ldots, n_k) \in S \iff \bigvee_{\varphi \in R} \varphi(n_1, \ldots, n_k)$. Note that $\exists$PAD sets are closed under union, intersection and projection.

It was shown in [Haa12, HKOW09] that the reachability relation of parametric one-counter machines is $\exists$PAD definable.

Lemma 2.7.1 ([Haa12], Lemma 4.2.2)**.** *Given a parametric one-counter machine C (i.e., no upper bounds, '$+[0, p]$' or '$\equiv 0 \bmod c$' transitions) and states s, t, the relation* $\mathrm{Reach}(C, s, t) = \{(x, y, n_1, \ldots, n_k) \mid (s, x) \to^* (t, y)$ *in* C^γ *where* $\gamma(p_i) = n_i\}$ *is* $\exists$PAD *definable.*

[4] A single formula would be logically sufficient, but would result in exponential blowup.



Chapter 3

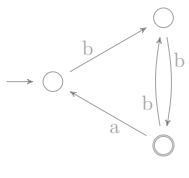
Completeness Thresholds in Bounded Model Checking

In this chapter we review how bounded model checking works and we show how to calculate completeness thresholds for given properties (given as finite or Büchi automata, respectively).

Given a model M and a property φ, the fundamental approach underpinning BMC is to look for counterexamples in M to φ of bounded length. As such, the absence of counterexample is inconclusive; a genuine bug could still lurk deeper within the system. For this reason, from the very inception of the technique, researchers have attempted to turn BMC into a *complete* method with the ability also to guarantee the absence of counterexamples of any length, e.g., [BCCZ99, CES08]. See also the work on cube enlargement techniques [McM02], circuit co-factoring [GGA04], induction [SSS00], and Craig interpolation [McM03].

For a property φ, we write $M \models_k \varphi$ if every k-bounded path satisfies φ. If $M \not\models \varphi$ then there is a smallest counterexample, a shortest path violating φ. A *completeness threshold* for M and φ is an integer k such that if $M \models_k \varphi$ then $M \models \varphi$. (This definition can be extended to infinite words by considering lasso-shaped k-bounded paths).

In [BCCZ99], Biere *et al.* observed that for simple safety properties of the form $\mathsf{G}p$, a *completeness threshold* is given by the *diameter* (longest distance between any two states) of the transition system under consideration: indeed, if no counterexample to $\mathsf{G}p$ of length at most the diameter of the system can be found, then no counterexample of any length can possibly exist. Likewise, for liveness properties such as $\mathsf{F}q$, the *recurrence diameter* (length of the longest loop-free path) of the transition system can be seen to be an adequate completeness threshold. See Figure 3.1 for the difference between diameter and recurrence diameter.



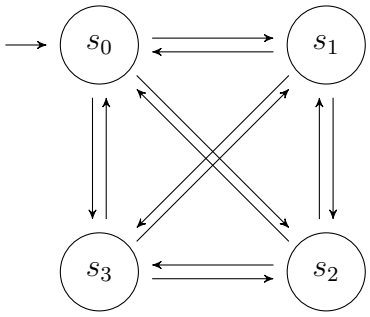


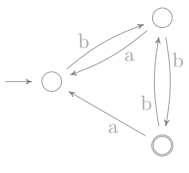
Figure 3.1: An example of a model M. Note that $d(M) = 1$ whereas $\mathrm{rd}(M) = 3$.

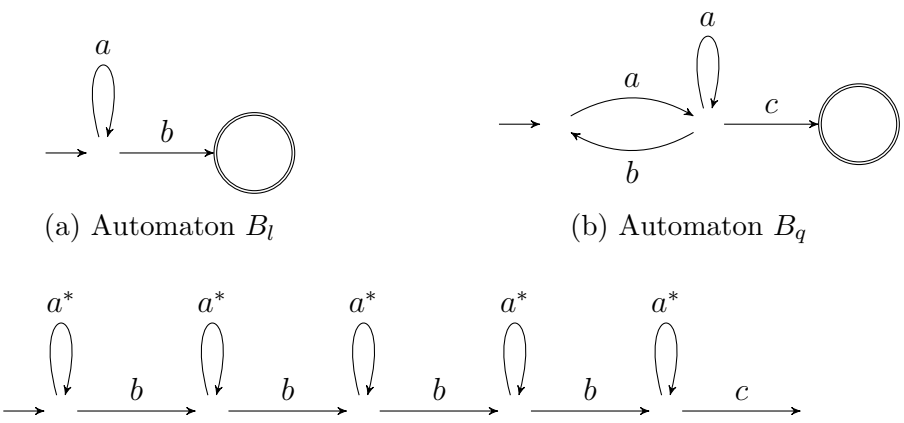
The notion of the completeness threshold is best stated in terms of automata. The completeness threshold for model M and property φ is the the length of the shortest accepting path in the product automaton $M \times B_{\neg\varphi}$ where $B_{\neg\varphi}$ is the automaton recognising the complement of φ.

For example, consider the two-state finite automaton B_l recognising the language a^*b (Figure 3.2). Let M be an arbitrary model and consider the shortest accepting path π in $M \times B_l$. Then $\mathrm{word}(\pi) = a^k b$ for some $k \in \mathbb{N}$. Thus, $\pi = (m_1, b_1) \to (m_2, b_1) \to \ldots \to (m_k, b_1) \to (m_{k+1}, b_2)$. for some states m_i of M. Since π is the shortest path, the prefix $\pi[1 \ldots k]$ visits every $m_1, \ldots, m_k$ at most once. Hence, $m_1 \to \ldots \to m_k$ is a simple path in M and hence $k \leq \mathrm{rd}(M)$. Therefore, $|\pi| = k+1 \leq \mathrm{rd}(M) + 1 \leq 2\,\mathrm{rd}(M)$. Since M was arbitrary, the length of the shortest accepting path in $M \times B$ is always at most twice the recurrence diameter of M. Thus, we say that B has linear completeness threshold.

On the other hand consider the finite automaton B_q and the model M_q in Figure 3.2. Observe that an accepting path in B_q cannot take two b edges in a row. Therefore, the shortest accepting path π in $M_q \times B_q$ is forced to visit every loop of M_q. Assuming M_q has n loops, each loop labelled by a^n, we have $\mathrm{sap}(M_q \times B_q) \geq n^2$. On the other hand, the longest loop-free path in M_q starts in the first loop, traverses the first loop almost entirely, then traverses the segment from the first to the last loop and then traverses the last loop almost entirely. Thus, $\mathrm{rd}(M_q) \leq 3n$. Since n can be arbitrary, the completeness threshold of B_q is at least quadratic in the recurrence diameter. In fact, by considering more intricate models we later show that the completeness threshold of B_q can be exponential in the recurrence diameter of models.

In a recent paper [KOS+11], Kroening *et al.* substantially extend the observations of Biere *et al.* [BCCZ99] by identifying a large class of ω-regular specifications for which completeness thresholds *linear* in the recurrence diameter of the models can be





(a) Automaton B_l (b) Automaton B_q

(c) Model M_q

Figure 3.2: A finite automaton with linear completeness threshold (a) nonlinear completeness threshold (b) and models witnessing nonlinearity (c).

effectively computed. This class consists of so-called *cliquey* (e.g., the automaton B_l in Figure 3.2 is cliquey.) Büchi automata, and subsumes among others the fragment of LTL consisting of unary next-free formulas. The authors also present examples of simple specifications having quadratic or even exponential completeness thresholds.[1]

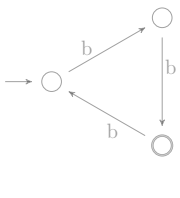
Unfortunately, [KOS+11] left as an open question whether the problem of determining if an ω-regular specification has a linear completeness threshold is decidable. In this chapter, we answer the question affirmatively by showing that the linearity problem for both regular and ω-regular specifications (provided as automata and Büchi automata respectively) is PSPACE-complete. Moreover, we establish the following dichotomies: for regular specifications, completeness thresholds are either linear or exponential, whereas for ω-regular specifications, completeness thresholds are either linear or at least quadratic (and can be provably quadratic and exponential).

3.1 Definitions

We now make the notions from the introduction formal. The convention we follow in this chapter is that models (transitions systems) are always denoted by M whereas automata are always denoted by B.

A state s of a transition system M is *reachable* if there is a path starting in the initial state and finishing in s. The *diameter* of M, denoted by $d(M)$, is the length of a longest shortest path between any two reachable states of M. The *recurrence*

[1]The precise definition of the magnitude of completeness thresholds is given in the next section.



diameter of M, denoted $\mathrm{rd}(M)$, is the length of a longest simple (loop-free) path through M. See Figure 3.1.

For an automaton B having at least one accepting path, we define $\mathrm{sap}(B) = \min\{k \mid B \text{ has a (lasso-shaped) } k\text{-bounded accepting path}\}$ to be the length of a *shortest accepting path.*

Note that a shortest path between any two states of B is necessarily simple. Hence $d(M) \leq \mathrm{rd}(M)$. By definition, every simple path visits each state of B at most once and hence $\mathrm{rd}(M) \leq |M|$. In case of finite automata, $\mathrm{sap}(B) \leq d(B)$ where we extend the definition of diameter to automata in the natural way.

Definition 3.1.1. *An automaton B has a **linear completeness threshold** if there exists $c \in \mathbb{R}^+$ such that for all models M, $\mathrm{sap}(M \times B) \leq c \cdot \mathrm{rd}(M)$.*

Definition 3.1.2. *An automaton B has **at least quadratic/exponential completeness threshold** if there exists a sequence of models $(M_i)_{i=1}^{i=\infty}$ with $\mathrm{rd}(M_i) \to \infty$ and a constant $c \in \mathbb{R}_+$ such that $\mathrm{sap}(M_i \times B) \geq c \cdot \mathrm{rd}(M_i)^2$ or $\mathrm{sap}(M_i \times B) \geq 2^{c \cdot \mathrm{rd}(M_i)}$, respectively.*

In short, we say that B is linear, at least quadratic or at least exponential. Finally, note that although we defined completeness thresholds using automata, it is a property purely of the *language* recognised by the automaton: B is linear if and only if for all models M the shortest word in $L(M) \cap L(B) \leq c \cdot \mathrm{rd}(M)$.

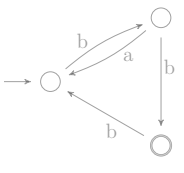
We also need the following notation. For states $(s_1, b_1), (s_2, b_2) \in M \times B$ let

$$\begin{aligned}\Pi^{(s_2,b_2)}_{(s_1,b_1)} = \{\pi \mid\ & \pi \text{ is a path through } M \times B, \\ & \mathrm{first}(\pi) = (s_1, b_1), \mathrm{last}(\pi) = (s_2, b_2), \\ & \forall i < |\pi| \,.\, \pi(i) \neq (s_2, b')\}\end{aligned}$$

be the set of paths through M starting in (s_1, b_1), ending in (s_2, b_2) and visiting the state s_2 only once. If ρ is a path in B such that $\mathrm{word}(\rho) = \mathrm{word}(\pi)$ then $\pi \otimes \rho$ denotes the unique path in $M \times B$ obtained by composing π and ρ componentwise.

3.2 Regular Languages

We begin by studying finite automata. For the rest of the section, let us a fix a finite automaton B. By the end of this section we show that nonlinearity of B is witnessed by models very much like the one depicted in Figure 3.2c, which we use to devise a decision procedure determining whether the completeness threshold of B is linear.



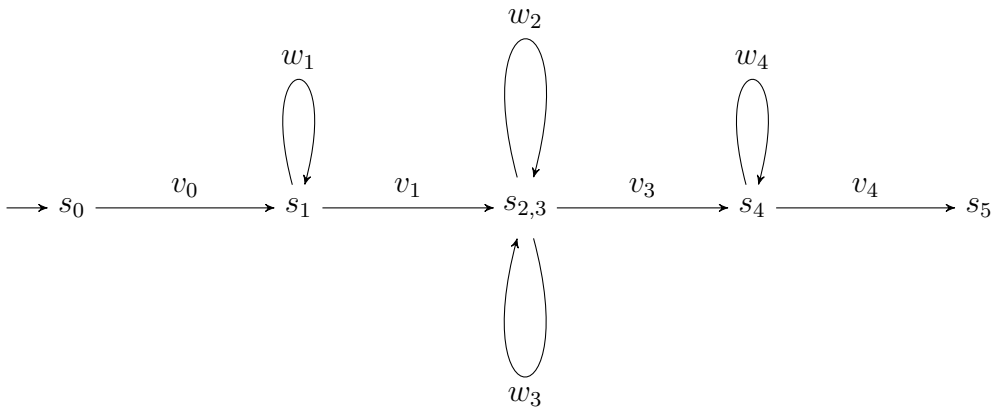


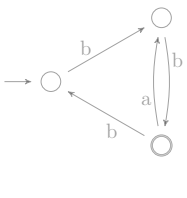
Figure 3.3: A loop model.

Note from the Definition 3.1.1 that to show that B is not linear, we have to find an infinite family of models M_i such that $\mathrm{sap}(M_i \times B) \geq i \,\mathrm{rd}(M)$. On the face of it, it is not clear that the existence of such an infinite family of models is even decidable. However, we show that the nonlinearity is always witnessed by a a special class of models. The class moreover possesses a small-model property and we show that there are arbitrarily large M_i's if and only if there is M_i for a large (but bounded) i.

Let M be a model. The structure of loops in M, as determined by the paths in the product $M \times B$, turns out to be crucial to linearity of B. For example, if π is a simple path in M, then it cannot possibly be longer than the recurrence diameter of M. In order to be longer than the recurrence diameter, π has to intersect itself and thus contain some loops. Roughly speaking, we will show that π needs approximately k reasonably well-behaved loops in order to be of length at least k times the recurrence diameter. These considerations motivate the introduction of models of the following special form, examples of which are shown in Figure 3.3 and Figure 3.2c

Definition 3.2.1. *Given $n \in \mathbb{N}$, words $w_1, \ldots, w_n \in \Sigma^+$ and words $v_0, v_1 \ldots, v_n \in \Sigma^*$ we define a **loop model** as follows. The model contains (among others) states $s_0, \ldots, s_{n+1}$. State s_0 is the only initial state. For every $0 \leq i \leq n$ there is a path, called an **arc**, from s_i to s_{i+1} that spells the word v_i and for $1 \leq i \leq n$ a loop path spelling w_i is attached to s_i. If $v_i = \epsilon$ then we identify s_i with s_{i+1}.*

As B is a finite automaton, it cannot distinguish between all words. We show that replacing loops and arcs by indistinguishable words in a loop model M does not affect M in any significant way. The notion of equivalent words and substitution of indistinguishable words is formalised in the following few definitions and technical lemmas.



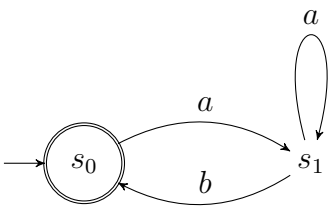


Figure 3.4: A finite automaton. Note that $R_{ab} = R_{aab} = \{(s_0, s_1), (s_1, s_0)\}$ and $S_{ab} = S_{aab} = \{(s_0, s_1)\}$.

A word w is a *prefix* of another word v if v can be written as $v = wx$ for some word x.

Definition 3.2.2. *Let $w \in \Sigma^*$. We define a transition relation $\boldsymbol{R_w}$ between states of B induced by w by $(b_1, b_2) \in R_w$ if $b_1 \xrightarrow{w} b_2$. And we define the set $\boldsymbol{S_w}$ of states of B such that $b \in S_w$ if $P(w) \cap L(b) \neq \emptyset$ where $P(w)$ is the set of all prefixes of w (including the empty string and w itself) and $L(b)$ is the set of words accepted by B starting in b.*

Informally, the pair (s, t) is in the relation R_w if there is a path starting in s, finishing in t and spelling w. The set S_w contain a state s of B if there is a path starting in s, spelling w and containing a final state. Equivalently, a state s of B is in S_w is there is a prefix w' of w and a path π starting in b ($\text{first}(\pi) = b$) such that the last state of π is a final state of B ($\text{last}(\pi) \in F$) and the word associated with π is w' ($\text{word}(\pi) = w'$).

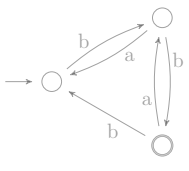
The automaton B then induces the following equivalence relation on Σ^*, cf. [Büc62].

Definition 3.2.3. *Two words $w, v \in \Sigma^+$ are equivalent, written $w \sim v$, if the corresponding relations and sets are equal, $R_w = R_v$ and $S_w = S_v$. We say that a word w or loop w is **pumpable** if the equivalence class $[w]$ is infinite. Denote the index of $\sim$ by C_B.*

For example, notice that for the automaton depicted in Figure 3.4, we have $ab \sim aab$. Further, the relation $\sim$ is a congruence, i.e., $w \sim v$ and $x \sim y$ imply that $wx \sim vy$.

Lemma 3.2.4. *The relation $\sim$ is a congruence. If v, w, x and $y \in \Sigma^*$ are words such that $w \sim v$ and $x \sim y$ then $wx \sim vy$.*

The proof of the lemma depends on the following two technical results.



Lemma 3.2.5. *Let $w, v \in \Sigma^*$ be words. Then $R_{wv} = R_v \circ R_w$.*

Proof. Suppose $(x, y) \in R_{wv}$. Then, there is a path $\pi : x \to y$ such that $\text{word}(\pi) = wv$. But then, π can be written as $\pi = \pi_1.\pi_2$ such that $\text{word}(\pi_1) = w$ and $\text{word}(\pi_2) = v$. Hence, $(x, last(\pi_1)) \in R_w$ and $(\text{first}(\pi_2), y) \in R_v$. Since $\text{last}(\pi_1) = \text{first}(\pi_2)$, we have $(x, y) \in R_v \circ R_w$.

Conversely, suppose that $(x, y) \in R_v \circ R_w$. Then there is z such that $(x, z) \in R_w$ and $(z, y) \in R_v$. Hence, there are paths $\pi_1 : x \to z$ and $\pi_2 : z \to y$ such that $\text{word}(\pi_1) = w$ and $\text{word}(\pi_2) = v$. It follows that $\pi_1\pi_2$ is well defined and $\text{word}(\pi_1\pi_2) = wv$ as required. □

Lemma 3.2.6. *Let $w, v \in \Sigma^*$ be words. Then $b \in S_{wv} \iff b \in S_w \vee \exists c \in S_v \,.\, (b, c) \in R_w$.*

Proof. Suppose $b \in S_w \vee \exists c \in S_v \,.\, (b, c) \in R_w$. If $b \in S_w$ then there exists a prefix w' of w such that $w' \in L(b)$. Then w' is a prefix of w as well and hence $b \in S_{wv}$.

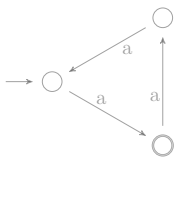
If $\exists c \in S_v \,.\, (b, c) \in R_w$ then there is a prefix v' of v such that $v' \in L(c)$. So there are paths $\pi_1 : b \to c$ and $\pi_2 : c \to d$ such that $\text{word}(\pi_1) = w$ and $\text{word}(\pi_2) = v'$ and $d \in F$ is a final state of B. But then $\pi_1\pi_2 : b \to d$ is a valid path starting in b and finishing in a final state and $\text{word}(\pi_1\pi_2) = wv'$ is a prefix of wv. Hence $b \in S_{wv}$.

Conversely, suppose that $b \in S_{wv}$. Then there is a path $\pi : b \to c$ for some final state $c \in F$ such that $\text{word}(\pi)$ is a prefix of wv. There are two cases. Either $\text{word}(\pi)$ is already a prefix of w in which case $b \in S_w$. In the second case, $\text{word}(\pi)$ is not a prefix of w. Hence, $\pi = \pi_1\pi_2$ such that $\text{word}(\pi_1) = w$ and $\text{word}(\pi_2)$ is a prefix of v. But now we can take c to be $c = \text{last}(\pi_1) = \text{first}(\pi_2)$. Then $(b, c) \in R_w$ and $c \in S_v$, which finishes the proof. □

Proof. (of Lemma 3.2.4) Suppose $R_w = R_v$ and $R_x = R_y$. Then by Lemma 3.2.5 we have $R_{wx} = R_x \circ R_w = R_y \circ R_v = R_{vy}$.

We also have $S_w = S_v$ and $S_x = S_y$. Hence, by Lemma 3.2.6 we also have that $b \in S_{wx} \iff b \in S_w \vee \exists c \in S_x.(b, c) \in R_w \iff b \in S_v \vee \exists c \in S_y.(b, c) \in R_v \iff b \in S_{vy}$ as required. □

Note that each S_w is a set of states of B. Hence, there are at most $2^{|B|}$ possible different values for S_w. Similarly, each R_w is a set of pair of states of B. Hence, there are at most $2^{|B|^2}$ possible different values for R_w. It follows, that the number C_B of equivalence classes of $\sim$ is at most $C_B \leq 2^{|B|^2+|B|}$. If the automaton B is clear from the context, we drop B and write C instead of C_B.



In general, the following lemma characterises pumpable words in terms of the number of equivalence classes of the underlying congruence.

Lemma 3.2.7. *Let $w \in \Sigma^*$ be a word, $\sim \in \mathcal{P}(\Sigma^* \times \Sigma^*)$ be a congruence and C be the number of equivalence classes of $\sim$.*

If w is not pumpable then $|w| \leq C$ and if w is pumpable then for any $K \in \mathbb{N}$ there is a word v such that $w \sim v$ and $K \leq |v| \leq K + C$.

Proof. Suppose $|w| > C$. Then, by pigeonhole principle, w decomposes as $w = xyz$ such that $x \sim xy$. Moreover, by taking the first such x and y, we can assume that $|xy| \leq C + 1$. Hence, $|y| \leq C$.

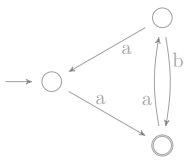
Now, a simple induction using the fact that $\sim$ is a congruence gives that $xy^k z \sim xyz$ for any $k \geq 0$. So w is pumpable. Moreover, as $|y| \leq C$, we have that for any $K \in \mathbb{N}$ there is a word v such that $w \sim v$ and $K \leq |v| \leq K + C$. □

3.2.1 Small-Model Property

Consider a loop model M. We shall show below that only the number of pumpable loops taken by accepting paths is essential to nonlinearity of B. Also, some of the loops of M may may not be taken by every accepting path and are thus redundant. We further restrict only to models without redundant loops and by concatenating loops attached to the same state we can also assume that at most one loop is attached to every state.

Definition 3.2.8. *Let $M = (n, \overrightarrow{w}, \overrightarrow{v})$ be a loop model with n loops such that $|v_i| > 0$ for $i > 0$ and let $\rho = ((m_1, b_1) \dots (m_t, b_t))$ be a path in $M \times B$. Then we can associate the vector $(x_1, \dots, x_n)$ with ρ where $x_i := |\{1 \leq j \leq t \mid m_j = s_i\}| - 1$ equals the number of times ρ takes the loop w_i (in the projection onto M). We say that ρ skips the loop w_i if $x_i \leq 0$. The model M is called an **irredundant loop model (for B)** if:*

- *$|v_i| > 0$ for every $0 < i \leq n$,*
- *every loop w_i is pumpable,*
- *no accepting path skips a loop,*
- *$L(M) \cap L(B) \neq \emptyset$.*



For example, the model M_q in Figure 3.2c is an irredundant loop model for the automaton B_q in Figure 3.2b.

We now state the main theorem of this chapter relating automata with nonlinear completeness threshold and irredundant loop models. The third statement forms the crucial part of our decision procedure.

Theorem 3.2.9. *Let B be an automaton. Then the following are equivalent.*

(a) B does not have a linear completeness threshold.

(b) For every $k \in \mathbb{N}$ there exists an irredundant loop model with at least k loops.

(c) There exists an irredundant loop model with L loops such that $2K \geq L > K$, where $K = 2^{2|B|}|B|$.

We first show that $(b) \implies (a)$. Then we prove that $(b) \iff (c)$ and justify the specific value of K. Finally, we establish that $(a) \implies (b)$.

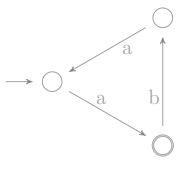
We begin by proving a stronger version of $(b) \implies (a)$. The result relies on the fact that replacing a loop by an equivalent one in an irredundant loop model keeps the model irredundant:

Lemma 3.2.10. *Let $M = (n, \overrightarrow{w}, \overrightarrow{v})$ be an irredundant loop model and let N be the model obtained by replacing w_i by x for some $x \sim w_i$ and $1 \leq i \leq n$. Then N is an irredundant loop model.*

Proof. Let π be an accepting path through $N \times B$ and suppose, to the contrary, that it skips some loop. We shall turn π into an accepting path ρ through $M \times B$ that takes the loop w_i as many times as π takes x and is the same as π elsewhere.

If $\pi[k \dots l]$ for $k < l$ is one traversal of x by π then $\pi(k) = (s_i, b_k)$ and $\pi(l) = (s_i, b_l)$. Thus $b_k \xrightarrow{x} b_l$ in B. Since $w_i \sim x$, it holds that $b_k \xrightarrow{w_i} b_l$. So there is a path τ in B from b_k to b_l. Let α be a path through M from s_i to s_i corresponding to one traversal of the loop w_i. Then replace $\pi[k \dots l]$ by $\alpha \otimes \tau$.

If π terminates inside the loop x then let $\pi(k)$ be the last occurrence of s_i in π. Then $\pi(k) = (s_i, b_k)$ and $\text{last}(\pi) = (m, b_l)$ for some m inside the loop x and b_l accepting state. Thus, $b_k \in S_x$ and since $w_i \sim x$, it holds that $b_k \in S_{w_i}$. So there is a path τ_2 in B from b_k to b_m for some accepting state b_m of B such that $\text{word}(\tau_2)$ is a prefix of w_i. Let α_2 be a path through M from s_i traversing the loop w_i and spelling $\text{word}(\tau_2)$. Then we replace $\pi[k \dots]$ by $\alpha_2 \otimes \tau_2$ thereby obtaining a path ending in a final state. □



By a similar proof we can show that replacing arcs by equivalent ones keeps the model irredundant.

Lemma 3.2.11. *Let $M = (n, \overrightarrow{w}, \overrightarrow{v})$ be an irredundant loop model and let N be the model obtained by replacing v_i by x for some $x \sim v_i$ and $0 \leq i \leq n$. Then N is an irredundant loop model.*

Proof. Let π be an accepting path through $N \times B$ and suppose, to the contrary, that it skips some loop. We shall turn π into an accepting path ρ through $M \times B$.

If $\pi[k \dots l]$ for $k < l$ is the traversal of x by π then $\pi(k) = (s_i, b_k)$ and $\pi(l) = (s_{i+1}, b_l)$. Thus $b_k \xrightarrow{x} b_l$ in B. Since $v_i \sim x$, it holds that $b_k \xrightarrow{v_i} b_l$. So there is a path τ in B from b_k to b_l. Let α be a path through M from s_i to s_{i+1} corresponding to one traversal of the arc v_i. Then replace $\pi[k \dots l]$ by $\alpha \otimes \tau$.

If π terminates after traversing just a prefix of x then let $\pi(k)$ be the last occurrence of s_i in π. Then $\pi(k) = (s_i, b_k)$ and $\text{last}(\pi) = (m, b_l)$ for some m inside the arc x and b_l accepting state. Thus, $b_k \in S_x$ and since $v_i \sim x$, it holds that $b_k \in S_{v_i}$. So there is a path τ_2 in B from b_k to b_m for some accepting state b_m of B such that $\text{word}(\tau_2)$ is a prefix of v_i. Let α_2 be a path through M from s_i traversing the arc v_i and spelling $\text{word}(\tau_2)$. Then we replace $\pi[k \dots]$ by $\alpha_2 \otimes \tau_2$ thereby obtaining a path ending in a final state. □

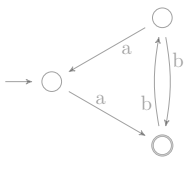
We can now prove a stronger version of $(b) \implies (a)$ in Theorem 3.2.9.

Theorem 3.2.12. *Let B be an automaton. If for every $k \in \mathbb{N}$ there exists an irredundant loop model with at least k loops then B's completeness threshold is at least quadratic.*

Proof. Let $M = (n, \overrightarrow{w}, \overrightarrow{v})$ be an irredundant loop model with n loops. Then, using Lemma 3.2.7, change every w_i to y_i such that $w_i \sim y_i$ and $n \leq |y_i| \leq n + C$. Also, change every v_i to x_i such that $v_i \sim x_i$ and $|x_i| \leq C$. Denote the obtained model by M'. Lemmas 3.2.10 and 3.2.11 guarantee that M' is irredundant. Since a longest loop-free path in M visits at most two loops and traverses all v_i's, we have $\text{rd}(M') \leq 2(n+C) + (n+1)C \leq 2nC + 2nC + nC + nC = 6nC$. On the other hand, $\text{sap}(M' \times B) \geq n^2$ as every accepting path traverses each of at least n loops and each loop is of length at least n. Thus,

$$\text{sap}(M' \times B) \geq n^2 \geq \frac{\text{rd}(M')^2}{36C^2}$$

Since n can be arbitrarily large, the result follows. □



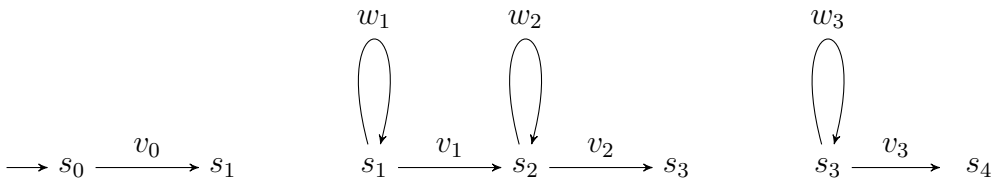


Figure 3.5: A partition of an irredundant loop model.

Next, we shall show that if there is an irredundant loop model with more loops than a certain critical threshold then there are irredundant loop models with arbitrarily many loops.

Let $M = (n, \overrightarrow{w}, \overrightarrow{v})$ be an irredundant loop model. Recall that we denoted the state of M to which the loop w_i is attached by s_i and the initial state of B by b_0. Then for every $0 < i < n$, the states of B can be assigned into the following two (possibly overlapping) categories.

$$E_i := \{b \in B \mid \Pi^{(s_i,b)}_{(s_0,b_0)} \neq \emptyset\}$$
$$F_i := \{b \in B \mid \exists \pi \in \Pi^{(s_i,b)}_{(s_0,b_0)} . \, \pi \text{ skips some } w_j \text{ for } j < i\}$$

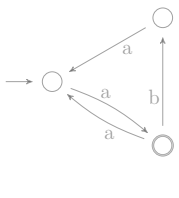
The set E_i collects the reachable states of B at the state s_i of M and the set F_i collects the states which are reachable by a path skipping a loop.

Furthermore, let π be an accepting path in $M \times B$. Then π visits every state $s_i \in M$ and so let $f(i)$ be the index of the first occurrence of s_i in π. If $n > 2^{2|B|}|B|$ then, by the pigeonhole principle, there are indices $1 \leq i < j \leq n$ such that $E_i = E_j, F_i = F_j$ and there is a state $b \in B$ such that $\pi(f(i)) = (s_i, b)$ and $\pi(f(j)) = (s_j, b)$.

Now, partition M into three loop models, corresponding to the prefix, pumpable segment and the suffix of M respectively $X := (i-1, (w_1, \ldots, w_{i-1}), (v_0, \ldots, v_{i-1}))$, $Y := (j-i, (w_i, \ldots, w_{j-1}), (\epsilon, v_i, \ldots, v_{j-1}), Z := (n-j+1, (w_j, \ldots, w_n), (\epsilon, v_j, \ldots, v_n))$. For example, the partition of an irredundant loop model with 3 loops split at $i = 1, j = 3$ is depicted in Figure 3.5.

Thus, M can be written as $X \to Y \to Z$ where the last state of one part is identified with the first state of the next part. Similarly, we partition the path π into subpaths in X, Y and Z. Let $\alpha = \pi[1 \ldots f(i)], \beta = \pi[f(i) \ldots f(j)]$ and $\gamma = \pi[f(j) \ldots)$. The following theorem then shows that Y can be pumped while keeping the model irredundant.

Lemma 3.2.13. *The model $M' = X \to Y \to Y \to Z$ is irredundant and $\alpha\beta\beta\gamma$ is an accepting path through $M' \times B$.*



Proof. By contradiction, making case distinction on the place where an accepting path skips a loop.

Let i, j be as above. Observe that the states s_i, s_j of M' correspond to the first state of the first copy of Y and the first state of the second copy of Y, respectively. Let ρ be an accepting path in $M' \times B$ and suppose, to the contrary, that ρ does not take some loop w_k of M'. If ρ does not visit the state s_j then ρ is completely contained in $(X \to Y) \times B$ and so ρ is an accepting path in $M \times B$ skipping w_n. So suppose that ρ visits the state s_j and let $\rho(f(j)) = (s_j, t)$ be the state of ρ when it is first visited. Denote the suffix of ρ starting at $f(j)$ by $\tau_2 = \rho[f(j)\ldots)$. Note that τ_2 is a path in $(Y \to Z) \times B$ that finishes in a final state. There are two possibilities either $k < j$ and w_k is in $X \to Y$ or $k \geq j$ and w_k is in $Y \to Z$.

Suppose that $1 \leq k < j$. Then $t \in F_j$ since t is reachable by a path that skips w_k. By the choice of Y, $t \in F_i$ as well and so there is a path τ_1 from the initial state to (s_i, t) in $M \times B$ that skips a loop. Note that τ_1 is a path in $X \times B$ and so $\tau_1\tau_2$ is an accepting path in $M \times B$ that skips some loop. But this is impossible as M is irredundant.

Now suppose that $j \leq k$. Then τ_2 is a path in $(Y \to Z) \times B$ that skips w_k. Also, t is reachable from the initial state and thus $t \in E_j$. By the choice of Y, $t \in E_i$ as well and so there is a path τ_1 from the initial state to (s_i, t) in $M \times B$. Note that τ_1 is a path in $X \times B$ and so $\tau_1\tau_2$ is an accepting path in $M \times B$ that does not take every loop. But this is impossible as M is irredundant.

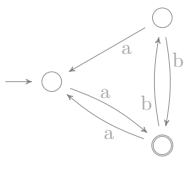
Finally, note that α, β and γ are paths through $X \times B$, $Y \times B$ and $Z \times B$ respectively. By the construction $\text{first}(\beta) = (s_i, b)$ and $\text{last}(\beta) = (s_j, b)$. Thus β concatenated with itself gives a valid path through $(Y \to Y) \times B$. Since γ ends in a final state, $\alpha\beta\beta\gamma$ is an accepting path in $M' \times B$. □

Similarly, by modifying the above proof slightly, we can also show that the Y part can be removed from M and the model remains irredundant.

Lemma 3.2.14. *The model $M'' = X \to Z$ is irredundant and $\alpha\gamma$ is an accepting path through $M'' \times B$.*

Proof. By contradiction, making case distinction on the place where an accepting path skips a loop.

Let i, j be as above. Observe that the state s_i of M' correspond to the last state of X. Let ρ be an accepting path in $M' \times B$ and suppose, to the contrary, that ρ does not take some loop w_k of M'. If ρ does not visit the state s_i then ρ is completely contained in $X \times B$ and so ρ is an accepting path in $M \times B$ skipping w_n. So suppose



that ρ visits the state s_i and let $\rho(f(i)) = (s_i, t)$ be the state of ρ when it is first visited. Denote the suffix of ρ starting at $f(i)$ by $\tau_2 = \rho[f(i)\ldots)$. Note that τ_2 is a path in $Y \times B$ that finishes in a final state. There are two possibilities either $k < i$ and w_k is in X or $k \geq i$ and w_k is in Z.

Suppose that $1 \leq k < i$. Then $t \in F_i$ since t is reachable by a path that skips w_k. By the choice of Y, $t \in F_j$ as well and so there is a path τ_1 from the initial state to (s_j, t) in $(X \to Y) \times B$ that skips a loop. So $\tau_1\tau_2$ is an accepting path in $M \times B$ that skips some loop. But this is impossible as M is irredundant.

Now suppose that $i \leq k$. Then τ_2 is a path in $Z \times B$ that skips w_k. Also, t is reachable from the initial state and thus $t \in E_i$. By the choice of Y, $t \in E_j$ as well and so there is a path τ_1 from the initial state to (s_j, t) in $(X \to Y) \times B$. So $\tau_1\tau_2$ is an accepting path in $M \times B$ that does not take every loop. But this is impossible as M is irredundant.

Finally, by construction, $\alpha\gamma$ is an accepting path in $M' \times B$. □

Since we can double as well as remove segments from large enough irredundant loop models, we have

Theorem 3.2.15. *There are irredundant loop models with arbitrarily many loops if and only if there is an irredundant loop model $M = (n, \overrightarrow{w}, \overrightarrow{v})$ with $2^{2|B|}|B| < n \leq 2^{2|B|+1}|B|$.*

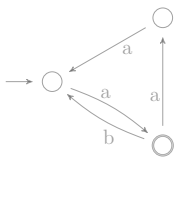
By flattening some of the repeated loops we shall prove in the following paragraphs that the nonlinearity of B is witnessed by very structured irredundant loop models.

Observe that the values of E_{i+1} and F_{i+1} depend only on E_i, F_i, w_i and v_i.

$$\begin{aligned} E_{i+1} &= \{b \mid \exists u \in E_i \,.\, (u \xrightarrow{w^k v} b \text{ for some } k \in \mathbb{N})\} \\ F_{i+1} &= \{b \mid \exists u \in F_i \,.\, (u \xrightarrow{w^k v} b \text{ for some } k \in \mathbb{N})\} \cup \\ &\quad \{b \mid \exists u \in E_i \,.\, (u \xrightarrow{v} b)\} \end{aligned}$$

Therefore, $E_i = E_j$ and $F_i = F_j$ even for $M' = X \to Y \to Y \to Z$. Thus the same reasoning can be applied inductively to pump Y thereby obtaining a family of irredundant loop models of the form $X \to Y^k \to Z$ for every $k \in \mathbb{N}$. The notation Y^k stands for concatenating Y models k times in a row: $Y^k = \underbrace{Y \to \cdots \to Y}_{k \text{ times}}$.

Theorem 3.2.16. *For every $k \in \mathbb{N}$ the model $X \to Y^k \to Z$ is irredundant and $\alpha\beta^k\gamma$ is an accepting path in $(X \to Y^k \to Z) \times B$.*



Next, we transform Y so that it contains only a single loop and a single arc. Suppose that π traverses the loop w_i exactly x_i times. Let $v := v_i w_{i+1}^{x_{i+1}} v_{i+1} \dots w_{j-1}^{x_{j-1}} v_{j-1}$ be the unwinding of all but the first loop of Y according to β. That is, $\text{word}(\beta) = w_i^{x_i} v$. Finally, define Y' to be the loop model $(1, w_i^{x_i}, (\epsilon, v))$. Observe that Y' is only a submodel of Y and since $X \to Y^k \to Z$ is irredundant, so is $X \to (Y')^k \to Z$. In particular, $\text{word}(\alpha\beta^k\gamma) \in L((X \to Y'^k \to Z) \times B)$. Further, by flattening the loops in X and Z we obtain a family of models of the form $x \to (Y')^* \to z$ where $x = \text{word}(\alpha)$ and $z = \text{word}(\gamma)$ (Model M_q in Figure 3.2c is of this form). Thus we have:

Theorem 3.2.17. *For every $k \in \mathbb{N}$, the model $N_k := x \to Y'^k \to z$ is irredundant.*

Proof. Notice that if $k = 0$ then N_k has no loops and so is trivially irredundant. So suppose that $k \geq 1$. Suppose, to the contrary, that there is a path $\pi \in N_k$ skipping a loop such that $\text{word}(\pi)$ is accepted by B, i.e., $\text{word}(\pi) \in L(B)$.

Denote the only remaining loop in Y' by w. By the construction of N_k we have that $\text{word}(\pi)$ is a prefix of a word of the form $xw^{a_1}vw^{a_2}v \dots w^{a_k}vy$ where a_p denotes the number of traversal of the loops w in the pth copy of Y'.

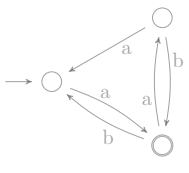
Since π skips some loop, we must have $a_q = 0$ for some $1 \leq q \leq k$. But then expanding the definition of w and v we have that $\text{word}(\pi)$ corresponds to a path in $X \to Y^k \to Z$ that skips the copy of w_i in the qth copy of Y. But this is impossible as by construction we have $\text{word}(\pi) \in L(N_k) \subseteq L(X \to Y^k \to Z)$ and the latter model is irredundant (Theorem 3.2.16).

So every accepting path in $N_k \times B$ takes every loop. Finally, by the reasoning in the paragraph above, we have that $\text{word}(\alpha\beta^k\gamma) \in L((X \to Y'^k \to Z) \times B)$. Thus, $L(N_k) \cup L(B) \neq \emptyset$ and so N_k is irredundant. □

Denote the only remaining loop in Y' by w. The models N_k's have very intricate structure. In particular, no accepting path through $N_k \times B$ takes two v's in succession or takes x immediately followed by v. Consider the fractal-like models as shown in the Figure 3.7. The model is obtained by identifying the leaves of two complete binary trees, orienting the edges as in the figure and by adding back edges from one tree to the other.

We define the fractal-like models inductively. First, we have to extend the definition of a model to allow an edge being labeled by a word from Σ^* (not only by a single letter from the alphabet Σ).[2]

[2]By expanding the edge into a sequence of edges according to individual letters in the word, we obtain a traditional model.



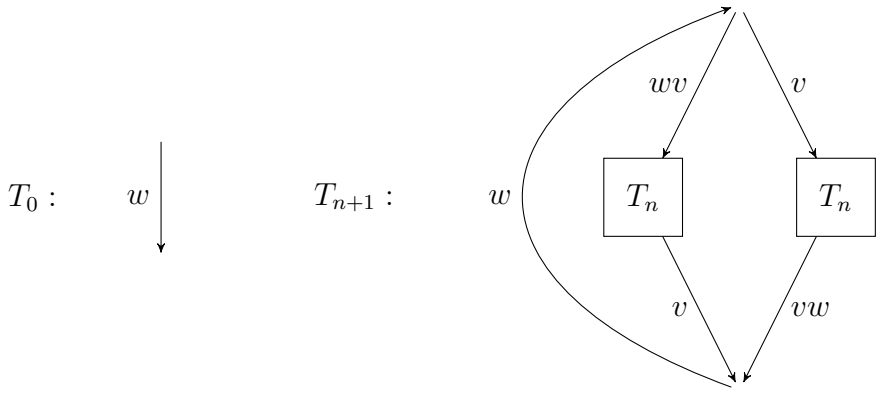


Figure 3.6: Inductive definition of models with the shortest accepting path exponential in the recurrence diameter.

The fractal-like models are defined as follows. For every $k \geq 0$ the model T'_k is defined inductively using the rules shown in Figure 3.6. Then the model T_k is obtained from T'_k by adding an edge labelled x leading to the topmost state of T_k and by adding an edge labelled vz leaving from the bottommost state of T_k. For example, the model depicted in Figure 3.7 is T_3. By *height* $h(k)$ of T'_k we denote the number of edges on a shortest path from the topmost state of T'_k to the bottommost state of T'_k.

The models T'_k satisfy the following properties. In particular, observe that the number of edges in T'_k and hence T_k is exponential in $\mathrm{rd}(T_k)$.

Lemma 3.2.18. *The models T'_k satisfy the following properties:*

- *Number of edges in T'_k equals $3 \cdot 2^{k+1} - 5$.*
- *The height of T'_k equals $h(k) = 2k + 1$.*
- $\mathrm{rd}(T'_k) \leq 4k + 3$.

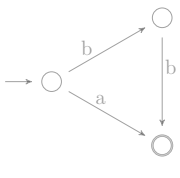
Proof. Denote the number of edges in T'_k by E_k. By induction on k we prove that $E_k = 3 \cdot 2^{k+1} - 5$.

For $k = 0$ note that (Figure 3.6) $E_0 = 1 = 3 \cdot 2^1 - 5$.

For the inductive step, suppose that $E_k = 3 \cdot 2^{k+1} - 5$. Then using the inductive definition (Figure 3.6), we have

$$E_{k+1} = 2E_k + 5 = 2(3 \cdot 2^{k+1} - 5) + 5 = 3 \cdot 2^{k+2} - 5$$

as required.



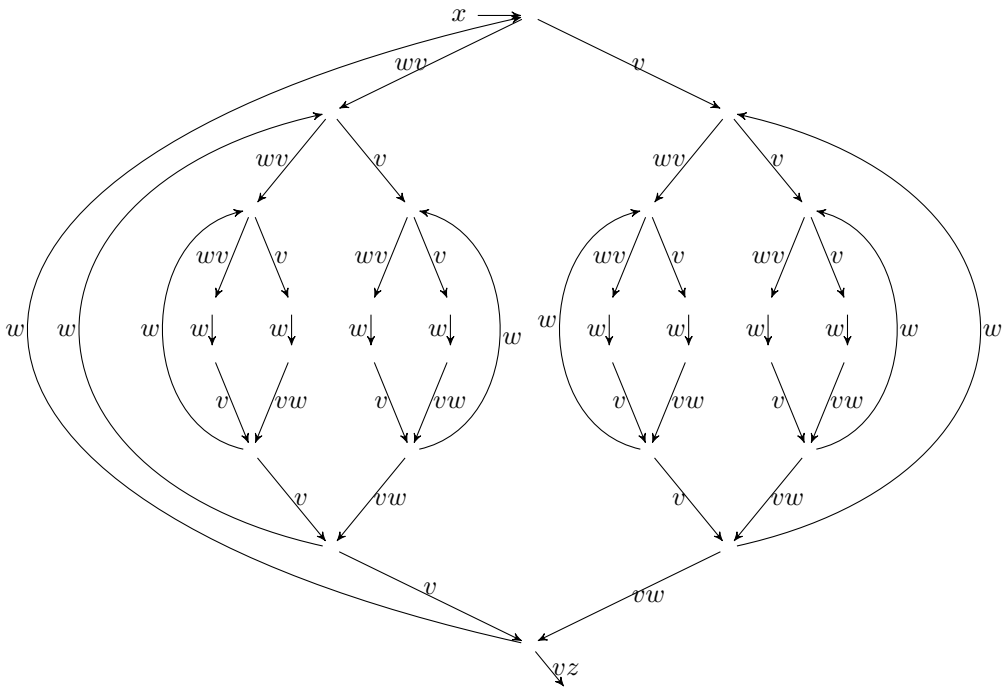


Figure 3.7: Model with the shortest accepting path exponential in the recurrence diameter.

We prove by induction that $h(k) = 2k + 1$. The case $k = 0$ follows directly from the inductive definition (Figure 3.6).

For the inductive step, suppose that $h(k) = 2k + 1$. Then using the inductive definition (Figure 3.6), we have

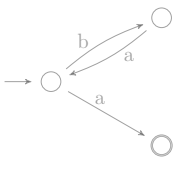
$$h(k+1) = h(k) + 2 = 2k + 1 + 3 = 2(k+1) + 1$$

as required.

Finally, note that Figure 3.6 uniquely defines the direction of each edge as going up or down. Consider a loop-free path π through T'_k. If π takes no edges going up then $|\pi| \leq h(k) = 2k + 1$.

Otherwise, observe that π can take only one edge going up. Therefore π can be written as $\pi = \pi_1 e \pi_2$ where e is an edge in the upward direction and all edges along π_1 and π_2 go downward. Hence $|\pi| = |\pi_1| + 1 + |\pi_2| \leq 2h(k) + 1 \leq 4k + 3$. □

So let ρ be an accepting path in $T_r \times B$ and m equals the number of traversals of v by ρ. Note that $\text{word}(\rho) = x(v|w)^* vz$ and hence ρ can be easily modified into an accepting path in $N_m \times B$ spelling the same word. Using the properties of N_m



it follows that ρ never takes two v's in a row. Notice that whenever any submodel T'_{k+1} is entered by ρ, the proceeding edge taken by ρ was labelled by v. Hence, in Figure 3.6 the first edge taken in T'_{k+1} by ρ is the wv edge going left.

Similarly, notice that whenever any submodel T'_{k+1} is exited by ρ, the edge taken by ρ starts with v. Therefore, in Figure 3.6 the last edge taken by ρ is the bottom-right edge labelled vw. So ρ visits both the left and the right branch of T'_{k+1} in Figure 3.6 and so it follows that ρ traverses the entire T_r.

The path ρ might stay in a submodel of M for a while, but eventually, it has to traverse edge vz and hence every edge of M. Thus, using Theorem 3.2.9, the family of models $T_1, T_2, T_3, \ldots$ witnesses that:

Theorem 3.2.19. *If the completeness threshold of an automaton B is not linear, then it is at least exponential.*

This result is also optimal. Let M be a general (not necessarily irredundant) model and let V be the number of its states. We show that $\mathrm{rd}(M)$ is always at least logarithmic in $\mathrm{sap}(M \times B)$.

In general, the shortest accepting path visits each state at most once. Hence $\mathrm{sap}(M \times B) \leq |M \times B| = |B|V$. Further observe that by removing an edge from M the recurrence diameter of M never increases whereas $\mathrm{sap}(M \times B)$ never decreases.

Lemma 3.2.20. *Let M be a model and M' be a model obtained from M by removing an edge. Then* $\mathrm{rd}(M) \geq \mathrm{rd}(M')$ *and* $\mathrm{sap}(M \times B) \leq \mathrm{sap}(M' \times B)$.

Proof. Let $P(M)$ denote the set of all paths in M. Then note that $P(M') \subseteq P(M)$. Hence, $P(M' \times B) \subseteq P(M \times B)$.

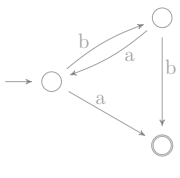
Now, the recurrence diameter is the longest loop-free path. As there are more paths in M as there are in M' we have $\mathrm{rd}(M) \geq \mathrm{rd}(M')$.

Similarly, sap denotes the shortest accepting path. As there are more paths in $M \times B$ as there are in $M' \times B$ we have $\mathrm{sap}(M \times B) \leq \mathrm{sap}(M' \times B)$. □

Therefore, we can assume that M contains only edges on the shortest accepting path. Thus the out-degree and in-degree of every node in M is at most $|B|$. Hence, the number of states reachable in K steps or less from the initial state is at least

$$\sum_{i=0,\ldots,K} |B|^k = \frac{|B|^{K+1} - 1}{|B| - 1} \leq |B|^{K+1}.$$

Since all V states of M are reachable, there must be a state at distance at least $\log_{|B|} V - 1$ from the initial state. Therefore, the diameter, and hence the



recurrence diameter, of M is at least $\mathrm{rd}(M) \geq \log_{|B|} V - 1 = \log_{|B|}(|B|V) - 2 \geq \log_{|B|}(\mathrm{sap}(M \times B)) - 2$.

Theorem 3.2.21. *Let M be a model such that $L(M \times B) \neq \emptyset$. Then* $\mathrm{rd}(M) \geq \log_{|B|}(\mathrm{sap}(M \times B)) - 2$.

3.2.2 From General Models to Irredundant Loop Models

We now prove the remaining implication $(a) \implies (b)$ of Theorem 3.2.9:

"Finite automaton B does not have a linear completeness threshold"
$\implies$
"For every $k \in \mathbb{N}$ there exists an irredundant loop model with at least k loops"

If B is nonlinear then there is an infinite family of arbitrary models witnessing it. In this section how to extract infinite family of irredundant models from the arbitrary models. Consider a model M. The main idea of the proof is to take a projection onto M of an accepting path through $M \times B$ and, by identifying non-overlapping loops, folding it into a loop model.

Let us demonstrate the main idea on an example. Let M be a (general) model and π be the projection onto M of an accepting path through $M \times B$. In general, π contains loops, which we use to define a loop model.

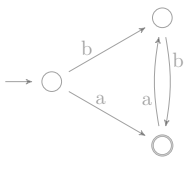
Suppose that the critical length of pumpable words $C = 8$, the length of the path $|\pi| = 80$ and the only loops in π are $\pi[10 \ldots 21], \pi[30 \ldots 40], \pi[40 \ldots 52]$ and $\pi[65 \ldots 75]$, i.e., the segments between these loops are loop-free.[3] In general, these loops may contain nested loops, but suppose for simplicity that they are simple. (See Figure 3.8).

Now, $\pi[40 \ldots 52]$ is a simple loop and thus $\mathrm{rd}(M) \geq 11$. Also, the path $\pi[52 \ldots 65]$ is loop-free and hence $\mathrm{rd}(M) \geq 13$. In general, such a split of π provides a bound on the recurrence diameter of M.

We use the loops induced by π to define a loop model. Let R be the loop model depicted in Figure 3.3, where we take $v_0 = \mathrm{word}(\pi[1 \ldots 10])$, $v_1 = \mathrm{word}(\pi[21 \ldots 30])$, $v_3 = \mathrm{word}(\pi[52 \ldots 65])$, $v_4 = \mathrm{word}(\pi[75 \ldots 80])$ and $w_1 = \mathrm{word}(\pi[10 \ldots 21])$, $w_2 = \mathrm{word}(\pi[30 \ldots 40])$, $w_3 = \mathrm{word}(\pi[40 \ldots 52])$ and $w_4 = \mathrm{word}(\pi[65 \ldots 75])$.

Observe that R embeds into M. For example, the path through R spelling $v_0 w_1^2 v_1 v_3 v_4$ corresponds to path $\pi[1 \ldots 10]\pi[10 \ldots 21]^2\pi[21 \ldots 30]\pi[52 \ldots 65]\pi[75 \ldots 80]$ through M. In general $L(R) \subseteq L(M)$. We use R, which is a simpler object, to study

[3] Event hough the segments are loop-free, they may still intersect each other (e.g., it is possible that $\pi(3) = \pi(25)$).



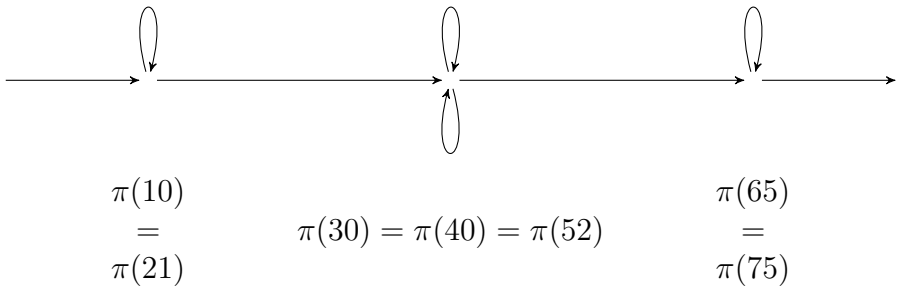


Figure 3.8: A structure of path π.

M. Observe that the length of every loop and arc in π is at most 13. Since loops are simple, we conclude that $13 \leq \mathrm{rd}(M)$.

Now, let ρ be the shortest accepting path through $R \times B$. Then ρ visits each state of R at most $|B|$ times. Since $\mathrm{word}(\rho) \in L(R) \subseteq L(M)$ the length $|\rho|$ provides an upper bound $\mathrm{sap}(M \times B) \leq |\rho|$. Precisely, ρ visits each arc (of which there are 4) and at most B loops from each loop bundle (of which there are 3). Hence $\mathrm{sap}(M \times B) \leq |\rho| \leq (4 + 3|B|)13 \leq 7|B| \,\mathrm{rd}(M)$.

Suppose that not only for this model but for every model M we can fold some accepting path into a loop model with 3 or fewer loop bundles. Then B is linear as in that case we can always bound $\mathrm{sap}(M \times B)$ by $7|B| \,\mathrm{rd}(M)$.

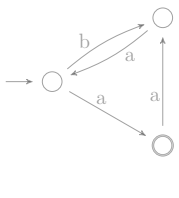
In general, suppose that there is $k \in \mathbb{N}$ such that for every model M we can fold some accepting path into a loop model with at most k loop bundles then B is linear and $\mathrm{sap}(M \times B) \leq 2(k+1)|B| \,\mathrm{rd}(M)$.

The contrapositive of the previous sentence reads: if B is nonlinear then for any $k \in \mathbb{N}$ there is a model such that every folding of every accepting path has more than k loop. And the smallest such folding, it turns out, can easily be modified into an irredundant loop model with approximately k loops.

In general, the loops and segments between loops are not loop-free but contain nested non-pumpable loops. In order to control the folding we need the loops in the projection as simple as possible. This motivates the following definition of locally minimal paths. Intuitively, a locally minimal path is loop-free upto at most C nonpumpable loops, each of which is of length at most C.

Definition 3.2.22. *A path ρ through M is **locally minimal** if,*

1. *Every state s of M appears at most C times in ρ.*



2. *The distance between successive occurrences of the same state is at most C. For $i < j$ we have if $\rho(i) = \rho(j)$ and $\forall i < k < j \,.\, \rho(k) \neq \rho(i)$ then $j - i \leq C$.*

We extend the definition to loops so that, up to the first and the last state which are the same, the rest of the path is locally minimal.

Definition 3.2.23. *A loop ρ of length n through M is **locally minimal** if,*

1. *If $\rho(i) = \rho(1)$ then $i = 1$ or $i = n$.*
2. *$\rho[2 \ldots n]$ is a locally minimal path.*

We can show that locally minimal paths and loops provide a bound on $\mathrm{rd}(M)$.

Lemma 3.2.24. *Let ρ be a locally minimal path through M. Then $\mathrm{rd}(M) \geq |\rho|/C^2$.*

Proof. Denote the length of ρ by n. We shall prove by induction on n that there is a loop-free path $\pi \subseteq \rho$ through M of length at least n/C^2. The path π starts in $\mathrm{first}(\rho)$.

The base case, $n = 1$, is trivial.

For $n > 1$, let $\rho(k)$ be the last occurrence of $\rho(1)$ in ρ. Possibly, $k = 1$. Since ρ is locally minimal, it holds that $k \leq C^2$. There are two cases:

- If $k = n$ then take $\pi = \rho(1)$. Then $|\pi| = 1$ and $n \leq C^2$.
- Otherwise, if $k < n$ then let π' be the path obtained by applying the induction hypothesis to $\rho[k+1, \ldots, n]$. Finally, take $\pi = \rho(1) \cdot \pi'$. Since π' begins with $\rho(k+1)$ and $\rho(1) = \rho(k)$, the sequence π is a path. Moreover, since $\rho(1) \notin \rho[k+1 \ldots)$, the path π is simple. Furthermore,

$$|\pi| = |\pi'| + 1 \geq \frac{n-k}{C^2} + 1 \geq \frac{n - C^2}{C^2} + 1 = \frac{n}{C^2}$$

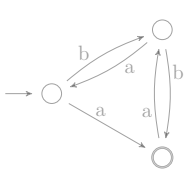
□

If ρ is a locally minimal loop then $\rho[2..|\rho|]$ is a locally minimal path. Hence, we have.

Lemma 3.2.25. *Let ρ be a locally minimal loop through M. Then $\mathrm{rd}(M) \geq (|\rho| - 1)/C^2$.*

We now define the formal notion of a folding of a path through M.

Definition 3.2.26. *Given:*



- *numbers* $k_1, \ldots, k_n > 0$,
- *locally minimal paths through* M: $\alpha_0, \ldots, \alpha_n$,
- *locally minimal loops through* M: $\beta_{1,1}, \ldots, \beta_{1,k_1}, \ldots, \beta_{n,1}, \ldots, \beta_{n,k_n}$

satisfying for all i, j:

- $\text{last}(\alpha_i) = \text{first}(\alpha_{i+1}) = \text{first}(\beta_{i+1,j}) = \text{last}(\beta_{i+1,j})$,
- $|\alpha_i| > 0$ *for* $0 < i < n$, *and*
- $|\beta_{i,j}| > C$,

a ***normalisation*** N *with* n *loop bundles is a loop model with arcs given by* $\text{word}(\alpha_0), \ldots, \text{word}(\alpha_n)$ *and with loops* $\text{word}(\beta_{i,1}), \ldots, \text{word}(\beta_{i,k_i})$ *in the* i*th loop bundle. A normalisation is called* ***accepting*** *if* $\text{first}(\alpha_0)$ *is the initial state of* M *and* $L(N \times B) \neq \emptyset$.

For example, we can think of the model in Figure 3.3 as a normalisation with 3 loop bundles of size $1, 2$ and 1, respectively. Observe that we can use locally minimal paths and loops to map every path through N to a corresponding path through M.

Lemma 3.2.27. *Let* N *be a normalisation. Then* $L(N) \subseteq L(M)$.

Proof. Let π by an accepting path in N. Then π is a prefix of the path of the form:

$$\alpha_0 \prod_{i=1,\ldots,n} (\beta_{i,1} \cup \cdots \cup \beta_{i,k_i})^*.$$

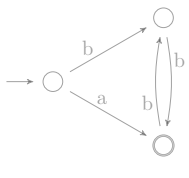
By construction of N, π is a valid path through M finishing in a final state and starting in $\text{first}(\alpha_0)$, which is the initial state of M. Hence, π is an accepting path through M. □

Then we use normalisations to bound $\text{sap}(M \times B)$ in terms of $\text{rd}(M)$.

Lemma 3.2.28. *Let* N *be an accepting normalisation with* n *loop bundles. Then* $\text{sap}(M \times B) \leq 4n|B|C^2 \,\text{rd}(M)$.

Proof. Consider the shortest accepting path π in $N \times B$. Then as π visits every state of $N \times B$ at most once, it follows that it visits each $s_i \in N$ at most $|B|$ times. Hence, π traverses at most $|B|$ loops in every loop bundle. Thus, using Lemmas 3.2.24 and 3.2.25 to bound the lengths of α's and β's, we obtain

$$\begin{aligned}
\text{sap}(M \times B) &\leq \text{sap}(N \times B) \\
&\leq (n+1)C^2 \,\text{rd}(M) + n|B|(C^2 \,\text{rd}(M) + 1) \\
&\leq 4n|B|C^2 \,\text{rd}(M).
\end{aligned}$$



The first term in the second line corresponds to arcs and the second term in the second line corresponds to loops. □

In particular, if there exists $k \in \mathbb{N}$ such that for every model M there is an accepting normalisation with fewer than k loop bundles then B is linear. It follows that

Theorem 3.2.29. *If B does not have a linear completeness threshold, then for every $k \in \mathbb{N}$ there is a model M_k such that $L(M_k \times B) \neq \emptyset$ and every accepting normalisation of M_k has at least k loop bundles.*

If $L(M \times B) \neq \emptyset$ then by instantiating the following result from the initial state to a reachable final state we obtain an accepting normalisation.

Theorem 3.2.30. *Let states $m_1, m_2 \in M$ and $w \in \Sigma^*$ be such that $m_1 \xrightarrow{w} m_2$. Then there exists a normalisation N such that*

- $|N| \leq |w|$,
- $\text{first}(\alpha_0) = m_1$,
- $\text{last}(\alpha_n) = m_2$,
- *there is a path π from the initial state of N to the last state of N such that* $\text{word}(\pi) \sim w$

where the α_i's refer to the locally minimal paths from Definition 3.2.26.

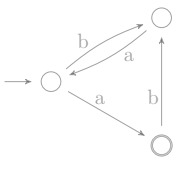
Proof. Let ρ be the shortest path in M from m_1 to m_2 such that $\text{word}(\rho) \sim w$. Clearly, $|\rho| \leq |w|$ and every state of M appears at most C times in ρ.[4]

We prove by induction on the length of ρ that there is a normalisation N such that $|N| \leq |\rho|, \text{first}(\alpha_0) = \text{first}(\rho), \text{last}(\alpha_n) = \text{last}(\rho)$ and there is a path π from the initial state of N to the last state of N such that $\text{word}(\rho) = \text{word}(\pi)$.

If ρ satisfies 2 from Definition 3.2.22 of locally minimal paths then we are done. Simply take $n = 0, \alpha_0 = \rho$. Otherwise, pick i, j with minimal j such that $\rho(i) = \rho(j)$, $i < j - C$ and $\forall i < k < j \,.\, \rho(k) \neq \rho(i)$. Then $\rho[i \ldots j]$ is a locally minimal loop.

If $i = 1$ then let N' be the normalisation obtained by applying the inductive hypothesis to $\rho[j \ldots)$. And add $\rho[1 \ldots j]$ into the first loop bundle of N' thereby obtaining N.

[4]Path ρ corresponds to the shortest path in the graph with vertices (s, c) where $v \in M$ is a vertex of M and c is an equivalence class of $\sim$.



If $i > 1$ then note that $\rho[1 \dots i]$ is a locally minimal path. Let N' be the normalisation obtained by applying the inductive hypothesis to $\rho[i \dots)$. And set $N = \rho[1 \dots i] \to N'$. □

Corollary 3.2.31. *Let M be a model such that $L(M \times B) \neq \emptyset$. Then an accepting normalisation exists.*

Finally, we show that the smallest accepting normalisation of M_k gives rise to an irredundant loop model with at least $k - 1$ pumpable loops.

Theorem 3.2.32. *Let B be nonlinear. Then for every $k \in \mathbb{N}$ there exists an irredundant loop model with at least $k - 1$ loops.*

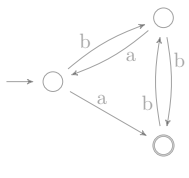
Proof. Let M_k be as defined above and let N be the smallest accepting normalisation of M_k. By the choice of M, N has at least k loop bundles. Recall that k_i's denote the number of loops in the i-th loop bundle. First, we show that every accepting path in $N \times B$ takes every loop at least once.

Suppose, to the contrary, that there is an accepting path π in $N \times B$ that skips some loop $w_{i,j}$. Let N' be obtained from N by removing the loop $w_{i,j}$ and the corresponding path $\beta_{i,j}$.

If $k_i > 1$ then N' is an accepting normalisation smaller than N. But this is impossible. If $k_i = 1$ then we have eliminated the i-th loop bundle from N thereby concatenating v_{i-1} with v_i and α_{i-1} with α_i. The concatenated path might not be locally minimal in M; however, we can replace it by a smaller normalisation thanks to Theorem 3.2.30. The net result is a compound accepting normalisation that is smaller than N, again yielding a contradiction. Thus every accepting path takes every loop.

Second, we show that it is possible to transform N so that every loop bundle is of size 1. Fix an accepting path π through $N \times B$ and recall that $w_{i_1}, \dots, w_{i_{k_i}}$ are the loops in the ith loop bundle of N. Let w be the word spelled by π while traversing through $w_{i_1}, \dots, w_{i_{k_i}}$ in $N \times B$. Then we remove $w_{i_1}, \dots, w_{i_{k_i}}$ from N and replace them by the single loop spelling w. By Definition 3.2.26 of a normalisation, each w_i is longer than C. Hence, $|w| > C$. By applying this transformation to every loop bundle, we obtain a new model N'. If $|\alpha_n| = 0$ then $v_n = \epsilon$. So let w_n be the last loop of N'. We remove w_n from N' and append the word w_n to v_{n-1} so that the last arc is non-null.

Note that word$(\pi) \in L(N')$. Since N' is just a submodel of N, every accepting path through $N' \times B$ traverses every loop. And so N' is an irredundant loop model. □



3.2.3 Decision Procedure

In the previous sections we showed that nonlinearity of a given finite automaton B is witnessed by a large enough irredundant loop model. In this section, we present a PSPACE decision procedure to determine whether a given automaton has a linear completeness threshold. If the automaton is linear we further show how to bound the linearity constant c from Definition 3.1.1. The section finishes by giving a corresponding proof of PSPACE-hardness.

Let $K = 2^{2|B|}|B|$. The decision procedure works by guessing a large enough loop model and then checking, on-the-fly, that the model is irredundant. Theorem 3.2.9 guarantees that to show nonlinearity of B it suffices to search for a loop model with number of loops between K and $2K$.

The decision procedure is shown in Algorithm 1. The algorithm first nondeterministically chooses the number of loops n in the model and then it keeps updating the set E_i of states reachable by a path in $M \times B$ and F_i of states reachable by a path skipping a loop. See the text before Lemma 3.2.13 on page 39 for the precise definition. To update these sets, the algorithm guesses a loop $w \in \Sigma^*$ and an arc $v \in \Sigma^*$. Instead of storing w and v, it calculates the relations R_w, R_v and sets S_w, S_v. These sets and relations are calculated by guessing the words letter by letter; using Lemmas 3.2.5 and 3.2.6 to update the sets incrementally.

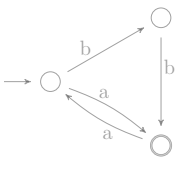
The algorithm needs to ensure that w is pumpable. However, Lemma 3.2.7 guarantees that it suffices to nondeterministically select a word of length between C and $2C$.

Note that only polynomially many bits in $|B|$ are needed to store R_w and S_w and since C and K are singly exponential in $|B|$, only polynomially many bits are needed to store n or the length of w. Also, only $|B|$ bits are needed to store E and F. Thus, the algorithm can be implemented in nondeterministic polynomial space. Finally, the algorithm ensures that a final state of B is visited only at the end.

We now prove a corresponding lower bound.

Theorem 3.2.33. *It is* PSPACE*-hard to determine whether an automaton B has a linear completeness threshold.*

Proof. The proof is by reduction from the universality problem for nondeterministic automata, well-known to be PSPACE-hard [HU79]. Let A be an automaton over alphabet Σ. We transform A into an automaton B over alphabet $\Sigma \cup \{\#\}$, where $\# \notin \Sigma$, such that A is universal if and only if B has a linear completeness threshold (see Figure 3.9).



Algorithm 1 Decision procedure for finite automata

$n \leftarrow$ guess a number $\in (K, 2K]$
$R_{v_0}, S_{v_0} \leftarrow$ guess a word
$E \leftarrow R_{v_0}(\{s_0\})$ ▷ s_0 is the initial state of B
$F \leftarrow \emptyset$.
for $i = 1$ to n **do**
  $R_w, S_w \leftarrow$ guess a pumpable word
  $R_v, S_v \leftarrow$ guess a word
  $E' \leftarrow \bigcup_{0 \le k \le |B|} R_{w^k v}(E)$
  $F' \leftarrow \bigcup_{0 \le k \le |B|} R_{w^k v}(F) \cup R_v(E)$
  if $S_{w^k} \cap E \neq \emptyset \vee (i < n \wedge \bigcup_{0 \le k \le |B|} S_{w^k v} \cap E \neq \emptyset) \vee (i = n \wedge S_v \cap F \neq \emptyset)$ **then**
    return false ▷ A final state is reachable before the end
  end if
  $(E, F) \leftarrow (E', F')$
end for
return $\big[(E \setminus F) \cap \text{Final}\big] \neq \emptyset$

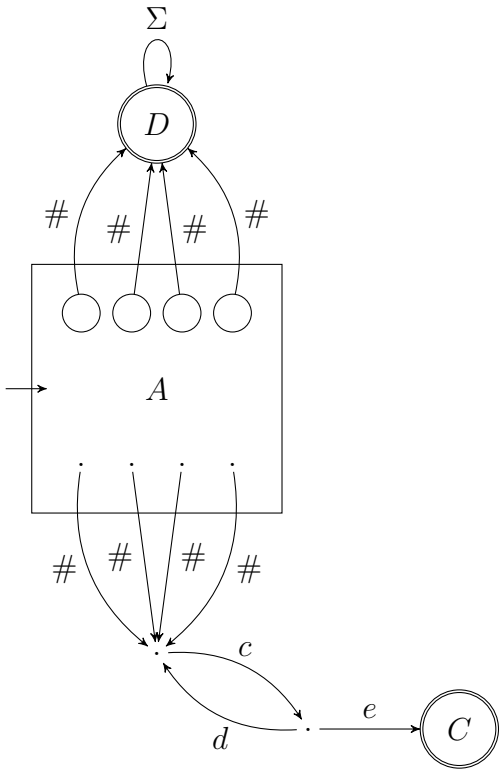
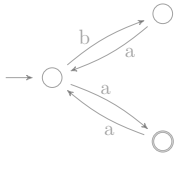


Figure 3.9: Automaton B used in the PSPACE-hardness proof.



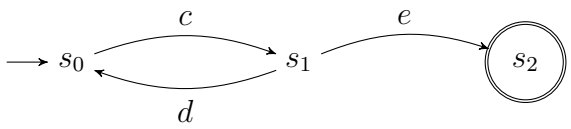


Figure 3.10: Automaton with exponential completeness threshold.

Let C be any automaton with non-linear completeness threshold (e.g. Figure 3.10) and denote the models witnessing nonlinearity by $(M_i)_{i\in\mathbb{N}}$. Let D be a single-state automaton accepting Σ^*. Define automaton B to be the disjoint union of A, C and D, with additional #-labelled transitions from each non-accepting state of A to the initial state of C and from each accepting state of A to the initial state of D. Then, make all states of A in B non-accepting. The construction of B can clearly be performed in polynomial time.

Clearly if A is universal then $L(B) = \Sigma^*\#\Sigma^*$. We claim that B has a linear completeness threshold in this case. Let M be a model such that $L(M \times B) \neq \emptyset$. Let $m \xrightarrow{\#} m'$ be a reachable # transition in M and let π be the shortest path from the initial state of M to m. By taking a prefix of π if necessary, we can assume that $\# \notin \mathrm{word}(\pi)$. Then $|\pi| \leq \mathrm{diam}(M) \leq \mathrm{rd}(M)$ and $\mathrm{word}(\pi)\# \in L(M) \cap L(B)$.

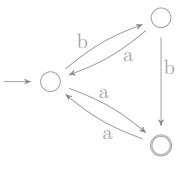
On the other hand, suppose that A does not accept some word $w \in \Sigma^*$. We claim that the models $(w\# \to M_i)_{i\in\mathbb{N}}$, obtained by attaching the last state of a path spelling $w\#$ to the initial state of M_i, witness nonlinearity of B.

Let π be an accepting path in $(w\# \to M_i) \times B$ and write $\pi = \pi_1\#\pi_2$. Since A rejects w, the #-transition in π must go from a rejecting state of A to the initial state of C. Thus π_2 is an accepting path in $M_i \times C$. Therefore $|\pi_2| \geq \mathrm{sap}(M_i \times C)$ and the claim follows since w is of a constant fixed size. □

3.2.4 Estimating the Linearity Constant

Suppose that we determined that B has a linear completeness threshold. Then there is a constant $c \in \mathbb{R}$ such that for every model M we have $\mathrm{sap}(M \times B) \leq c \cdot \mathrm{rd}(M)$. We call the infimum of all such constants the *linearity constant* of B.

If B is linear then by Theorem 3.2.15 we know that every irredundant loop model has at most K loops and so by instantiating Algorithm 1 for $n = 0, \ldots, K$ we can calculate the maximum number L of loops in an irredundant loop model. We claim that every model M has an accepting normalisation with at most $L+1$ loop bundles. Suppose not, then there is a model M_{L+2} such that every accepting normalisation of M_{L+2} has at least $L + 2$ loop bundles. But then by the argument in the proof



of Theorem 3.2.32 there is an irredundant loop model with $L+1$ loops, which is impossible. Thus, every model M has an accepting normalisation with at most $L+1$ loop bundles. Applying Lemma 3.2.28, we get that $\mathrm{sap}(M \times B) \leq 4(L+1)|B|C^2 \,\mathrm{rd}(M)$ thereby bounding the linearity constant.

As noted above, if B is linear, then every every irredundant loop model has at most K loops. Hence, every accepting normalisation of every model has at most $K+1$ loop bundles. Now, K depends only on $|B|$ and so even without calculating L we have that

Theorem 3.2.34. *If B is linear then the linearity constant c of B is bounded by $c \leq 4(K+1)|B|C^2$.*

3.3 ω-Regular Languages

We now extend the results from regular languages to ω-regular languages, from finite automata to Büchi automata. For the rest of the section, fix B to be a Büchi automaton. We show how to determine whether B has linear completeness threshold.

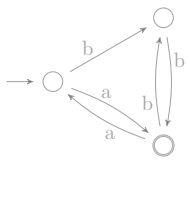
A word accepted by a Büchi automaton B is always infinite. Therefore, defining $\mathrm{sap}(B)$ to be the length of the shortest accepting word is meaningless. However, it is well-known (Lemma 2.2.11) that if there is an accepting path in B then there is a lasso-shaped one. Thus, for a Büchi automaton B we set $\mathrm{sap}(B) = k$ if the shortest lasso-shaped accepting path in B is k bounded:

$$\mathrm{sap}(B) = \min\{|w| + |v| \ : \ wv^\omega \in L(B)\}.$$

Also, implementations of bounded model checking translating the problem into propositional satisfiability look for k-bounded lasso-shaped paths.

The theory behind ω-regular languages is more technical than the theory of ordinary regular languages. However, almost all results in this section are obtained by adapting the arguments from finite case to infinite settings. The equivalent of an irredundant loop model that is suitable for ω-regular languages is depicted in Figure 3.11. The noose consists of one big loop with several (Figure 3.11) nested loops.

Definition 3.3.1. *An ω-loop model is specified by an integer n, words $w_1 \dots w_n$, $v_0 \dots v_n \in \Sigma^+$ and state s. The model contains (among others) states $s_0 \dots s_{n+1}$. State s_0 is the only initial state. For every $0 \leq i \leq n$ there is a path from s_i to s_{i+1} that spells out the word v_i and for $1 \leq i \leq n$ a loop path spelling w_i is attached to s_i. The state s lies on one of the paths spelling v_i and the state s_{n+1} is identified with s.*



3.3.1 Word Equivalence

Analogously to regular languages, we define equivalence on words suitable for replacing loops and arcs in irredundant ω-loop models. In case of regular languages, a path is accepting if it ends in a final state. On the other hand, for ω-regular languages an accepting path visits a final state infinitely often. Therefore, we modify the equivalence of two words to reflect this.

Definition 3.3.2. *Given $w \in \Sigma^*$ we define a binary relation $\boldsymbol{T_w}$ on B by $(b_1, b_2) \in T_w$ if there is a path ρ through B that begins in b_1, finishes in b_2, visits a final state and spells w.*

The relation is then used to define an equivalence on finite words.

Definition 3.3.3. *Given $w, v \in \Sigma^*$, the words are equivalent, written as $w \sim v$, if $R_w = R_v$ and $T_w = T_v$.*

As in the case of regular languages, the equivalence $w \sim v$ is a congruence.

Lemma 3.3.4. *Equivalence $w \sim v$ is a congruence.*

Proof. Let $w, v, x, y \in \Sigma^*$ be words such that $w \sim v$ and $x \sim y$. We need to show that $wx \sim vy$.

By Lemma 3.2.4 on page 34, we have that $R_{wx} = R_{vy}$.

By assumption, $R_w = R_v, R_x = R_y$ and $T_w = T_v, T_x = T_y$. Substituting into the lemma bellow, we have that

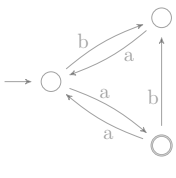
$$\begin{aligned}
(b_1, b_2) \in T_{wx} &\iff \exists c. R_w(b_1, c) \wedge R_x(c, b_2) \wedge (T_w(b_1, c) \vee T_x(c, b_2)) \\
&\iff \exists c. R_v(b_1, c) \wedge R_y(c, b_2) \wedge (T_v(b_1, c) \vee T_y(c, b_2)) \\
&\iff (b_1, b_2) \in T_{vy}
\end{aligned}$$

Hence $T_{wx} = T_{vy}$ as required. □

The previous lemma depends on the following technical result.

Lemma 3.3.5. *Let $w, v \in \Sigma^*$ be words and $b_1, b_2 \in B$ be states. Then*

$$(b_1, b_2) \in T_{wv} \iff \exists c. R_w(b_1, c) \wedge R_v(c, b_2) \wedge (T_w(b_1, c) \vee T_v(c, b_2)).$$



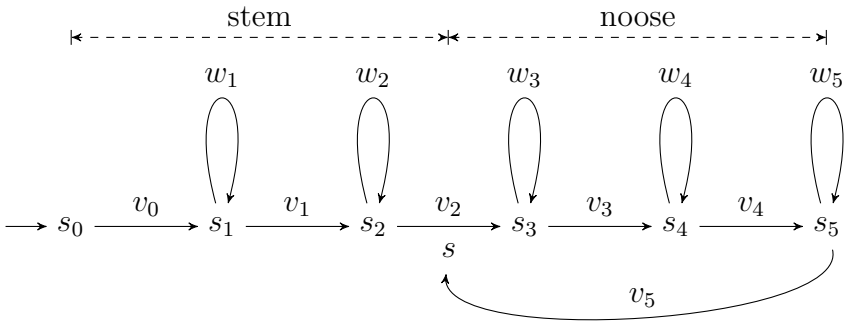


Figure 3.11: General form of a non linear model for ω-regular languages.

Proof.

$$
\begin{aligned}
(b_1, b_2) \in T_{wv} \quad &\iff \text{There is a path } \rho \text{ through } B \text{ such that, } \mathrm{first}(\rho) = b_1, \mathrm{last}(\rho) = b_2, \mathrm{word}(\rho) = wv \text{ and } \rho \text{ visits a final state.} \\
&\iff \text{There are paths } \rho_1, \rho_2 \text{ such that } \mathrm{first}(\rho_1) = b_1, \mathrm{last}(\rho_1) = \mathrm{first}(\rho_2), \mathrm{last}(\rho_2) = b_2, \mathrm{word}(\rho_1) = w, \mathrm{word}(\rho_2) = v \text{ and } \rho_1 \text{ or } \rho_2 \text{ visits a final state.} \\
&\iff \exists c. R_w(b_1, c) \wedge R_w(c, b_2) \wedge (T_w(b_1, c) \vee T_v(c, b_2))
\end{aligned}
$$

□

Similarly to regular languages, a word is *pumpable* if its equivalence class is infinite. Recall Lemma 3.2.7 relating pumpable words and the number of equivalence classes of B.

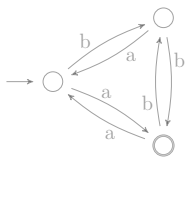
3.3.2 Characterisation of Nonlinear Büchi Automata

In this section, we state the main theorem characterising nonlinear Büchi automata. We say that a ω-loop model M is irredundant if

- $L(M \times B) \neq \emptyset$ there is an accepting path in $M \times B$,
- every loop in M is pumpable,
- for every accepting lasso-shaped path $\pi_1 \pi_2^\omega$ it holds that π_1 traverses every loop of the stem and π_2 traverses every loop of the noose.

Note that π_2 may traverse the noose more than once and in general, a single traversal of the noose might not traverse all the loops inside the noose.

Formally, let π be a lasso-shaped accepting path. Then $\pi = \pi_1 \pi_2^\omega$. By unwinding π_2 if necessary, we can assume that $\mathrm{first}(\pi_2) = \mathrm{last}(\pi_2) = s$. Recall, that for a state s



and a finite path τ, the expression $\text{count}(\tau, s)$ denotes the number of times the state s is visited by τ. So, $\text{count}(\pi_2, s) - 1$ denotes the number of traversals of the noose and hence $\text{count}(\pi_2, s_i) - (\text{count}(\pi_2, s) - 1)$ denotes the numbers of traversals of the loop s_i. Denote this number by k_i. An ω-loop model is irredundant if for every accepting lasso-shaped path π in $M \times B$ we have $k_i \geq 1$ for every i.

Finally, note that the shortest lasso-shaped accepting path takes at most $|B|$ traversal through the noose.

Theorem 3.2.9 on the relationship between finite automata with nonlinear completeness threshold and irredundant loop models has a direct counterpart in terms of Büchi automata and irredundant ω-loop models.

Theorem 3.3.6. *Let B be a Büchi automaton. Then the following are equivalent.*

(a) B does not have a linear completeness threshold.

(b) For every $k \in \mathbb{N}$ there exists an irredundant ω-loop model with at least k loops.

(c) There exists an irredundant ω-loop model with L loops such that $2K_\omega \geq L > K_\omega$, where $K_\omega = 2 \cdot 2^{4|B|^2}$.

Similarly to regular languages, we prove that condition (b) is equivalent to condition (a) and condition (c).

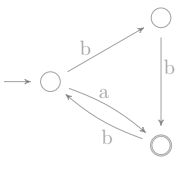
First, by pumping pumpable loops we show the equivalent of Theorem 3.2.12 that irredundant ω-loop models of arbitrary size witness nonlinearity of B. The result again depends on the fact that replacing a loop by an equivalent one in an irredundant ω-loop model keeps the model irredundant. Formally,

Lemma 3.3.7. *Let M be an irredundant ω-loop model and x be a word equivalent $x \sim w_i$ to some loop w_i of M. Then the model M' obtained from M be replacing w_i by x is an irredundant ω-loop model.*

Proof. Suppose that there is a an accepting path π in $M' \times B$ that skips a some loop l. If $l = x$ then π is an accepting path in $M \times B$, which is impossible. Hence $l = w_j$ for some $j \neq i$.

Now, let $\pi = \pi_1 \alpha \pi_2$ where α be some traversal of x. By construction, α can be replaced by a traversal β of the loop w_i such that $\text{first}(\alpha) = \text{first}(\beta)$ and $\text{last}(\alpha) = \text{last}(\beta)$. Moreover, we can assume that β visits a final state if and only if α does. This yields a path $\pi_1 \beta \pi_2$.

Replacing all traversals of x by traversals of w_i yields an accepting path in $M \times B$ skipping l. Contradiction. □



Similarly, the analogous result holds for arcs.

Lemma 3.3.8. *Let M be an irredundant ω-loop model and x be a word equivalent $x \sim v_i$ to some arc v_i of M. Then the model M' obtained from M be replacing v_i by x is an irredundant ω-loop model.*

We can now prove the stronger version of $(b) \implies (a)$ from Theorem 3.3.6.

Theorem 3.3.9. *If for every $k \in \mathbb{N}$ there exists an irredundant ω-loop model with at least k loops then B has at least quadratic completeness threshold.*

Proof. The proof is almost exactly the same as in the regular case (Theorem 3.2.12).

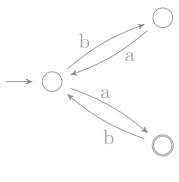
Let M be an irredundant ω-loop model with n loops. Then, using Lemma 3.2.7, change every loop w_i to y_i such that $w_i \sim y_i$ and $n \leq |y_i| \leq n + C$. Also, change every v_i to x_i such that $v_i \sim x_i$ and $|x_i| \leq C$. Denote the obtained model by M'. Lemmas 3.3.7 and 3.3.8 guarantee that M' is irredundant. Since a longest loop-free path visits at most two loops and traverses all v_i's, we have $\mathrm{rd}(M') \leq 2(n + C) + (n + 1)C \leq 2nC + 2nC + nC + nC = 6nC$. On the other hand, $\mathrm{sap}(M' \times B) \geq n^2$ as every accepting path traverses each of at least n loops and each loop is of length at least n. Thus,

$$\mathrm{sap}(M' \times B) \geq n^2 \geq \frac{\mathrm{rd}(M')^2}{36C^2}$$

Since n can be arbitrarily large, the result follows. □

The above theorem showed the implication $(b) \implies (a)$ from Theorem 3.3.6. Recall that to show the analogous statement of $(b) \iff (c)$ we proved a "small-model property" of irredundant models by carefully splitting the given irredundant loop model into three pieces $X \to Y \to Z$ so that the models $X \to Y^k \to Z$ are irredundant for every k. We proceed similarly for irredundant ω-loop models.

Suppose that M is an irredundant ω-loop model and M has more than K_ω loops. Then there are two possibilities. Either most of the loops are in the stem of M or in the noose of M. In the following sections we prove a small-model property separately for the two cases. In the former case, we use the techniques developed for finite languages to increase the number of loops. In the latter case, we extend the methods to pump the noose. We first show the former.



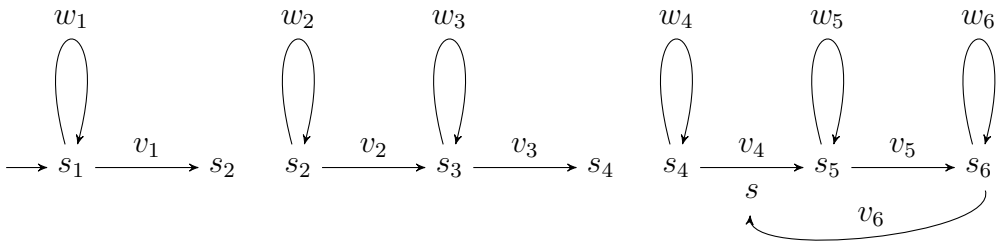


Figure 3.12: Splitting of an irredundant ω-loop model. Stem-heavy case.

3.3.3 Small-Model Property: Stem-Heavy Case

Throughout this subsection, let M be an irredundant ω-loop model such that at least half of its loops are in the stem. We show how to split M into three pieces $X \to Y \to Z$ such that $X \to Y^k \to Z$ is an irredundant ω-loop model for every $k \in \mathbb{N}$.

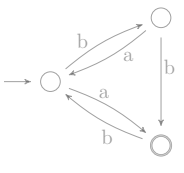
Recall (page 39) that for every s_i, the set E_i denoted the states of B reachable at s_i and the set $F_i \subseteq E_i$ consisted of states reachable by a path skipping a loop in $M \times B$. Consider such sets for every s_i in the stem.

Furthermore, let $\pi = \pi_1 \pi_2^\omega$ be an accepting path in $M \times B$. Then π visits every state $s_i \in M$ and so let $f(i)$ be the index of the first occurrence of s_i in π. If $n > 2^{2|B|}|B|$ then, by the pigeonhole principle, there are indices $1 \leq i < j \leq n$ such that $E_i = E_j, F_i = F_j$ and there is a state $b \in B$ such that $\pi(f(i)) = (s_i, b)$ and $\pi(f(j)) = (s_j, b)$.

Now, partition M (see Figure 3.12) into two loop models and one ω-loop model, corresponding to the prefix, pumpable segment and the suffix (with the noose) of M respectively as follows: $X := (i-1, (w_1, \ldots, w_{i-1}), (v_0, \ldots, v_{i-1}))$, $Y := (j - i, (w_i, \ldots, w_{j-1}), (\epsilon, v_i, \ldots, v_{j-1})$, $Z := (n - j + 1, (w_j, \ldots, w_n), (\epsilon, v_j, \ldots, v_n), s)$. The state s in Z denotes the state where the stem and the noose meet.

Thus, M can be written as $X \to Y \to Z$ where the last state of one part is identified with the first state of the next part. Similarly, we partition the path π into subpaths in X, Y and Z. Let $\alpha = \pi_1[1 \ldots f(i)], \beta = \pi_1[f(i) \ldots f(j)]$ and $\gamma = \pi_1[f(j) \ldots)$. The following theorem then shows that Y can be pumped while keeping the model irredundant.

Theorem 3.3.10. *The model $M' = X \to Y \to Y \to Z$ is irredundant and $\pi' = \alpha\beta\beta\gamma\pi_2^\omega$ is an accepting path.*



Proof. Note that $\alpha\beta\beta\gamma\pi_2^\omega$ is a valid path in M'. Since π is an accepting path the subpath π_2 visits a final state of B. Hence, π' visits a final state of B infinitely often and so π' is an accepting path.

We now show that M' is irredundant. Suppose, to the contrary, that it is not. Then there is an accepting path τ in $M' \times B$ that skips some loop w of M'. Let s_i and s_j be the first states of the first and the second copy of Y in M', respectively.

If τ is completely contained in $(X \to Y) \times B$ then τ is also an accepting path in $M \times B$ which skips a loop. But this is impossible as M is irredundant.

Therefore, τ reaches the state s_j and we can write τ as $\tau = \tau_1\tau_2\tau_3^\omega$ where τ_1 is the maximal prefix of τ completely contained in $(X \to Y) \times B$. Hence, $\tau_2\tau_3^\omega$ is a path in $(Y \to Z) \times B$ that visits a final state infinitely often. Further, denote the last state of τ_1 by $\text{last}(\tau_1) = \text{first}(\tau_2) = (s_j, t)$ for some state t of B. There are two cases.

Suppose τ_1 skips some loop w. Since (s_j, t) is reachable in $(X \to Y) \times B$ by a path that skips w, we have $t \in F_j$. By construction, $t \in F_j = F_i$ and so there is a path τ_1' in $X \times B$ that skips a loop and ends in (s_i, t). But then $\tau_1'\tau_2\tau_3^\omega$ is an accepting path in M that skips a loop, which is impossible as M is irredundant.

On the other hand, suppose that $\tau_2\tau_3^\omega$ skips a loop. Now, (s_j, t) is reachable in $(X \to Y) \times B$ and so we have $t \in E_j$. By construction, $t \in E_i$ as well and so there is a path τ_1'' in $X \times B$ that ends in (s_i, t). But then $\tau_1''\tau_2\tau_3^\omega$ is an accepting path in M that skips a loop, which is again impossible as M is irredundant. □

Proceeding by induction, we have

Theorem 3.3.11. *The model $M' = X \to Y^k \to Z$ is irredundant and $\alpha\beta^k\gamma\pi_2^\omega$ is an accepting path for any $k \geq 1$.*

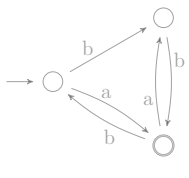
Similarly, we also have:

Theorem 3.3.12. *The model $M' = X \to Z$ is irredundant and $\alpha\gamma\pi_2^\omega$ is an accepting path.*

Proof. Note that $\alpha\gamma\pi_2^\omega$ is a valid path in M'. Since π is an accepting path the subpath π_2 visits a final state of B. Hence, π' visits a final state of B infinitely often and so π' is an accepting path.

We now show that M' is irredundant. Suppose, to the contrary, that it is not. Then there is an accepting path τ in $M' \times B$ that skips some loop w of M'. Let s_i be the last state of X in M' and s_j be the last state of Y in M, respectively.

If τ is completely contained in $X \times B$ then τ is also an accepting path in $M \times B$ which skips a loop. But this is impossible as M is irredundant.



Therefore, τ reaches the state s_i and we can write τ as $\tau = \tau_1\tau_2\tau_3^\omega$ where τ_1 is the maximal prefix of τ completely contained in $X \times B$. Hence, $\tau_2\tau_3^\omega$ is a path in $Z \times B$ that visits a final state infinitely often. Further, denote the last state of τ_1 by $\text{last}(\tau_1) = \text{first}(\tau_2) = (s_i, t)$ for some state t of B. There are two cases.

Suppose τ_1 skips some loop w. Since (s_i, t) is reachable in $X \times B$ by a path that skips w, we have $t \in F_i$. By construction, $t \in F_i = F_j$ and so there is a path τ_1' in $(X \to Y) \times B$ that skips a loop and ends in (s_j, t). But then $\tau_1'\tau_2\tau_3^\omega$ is an accepting path in M that skips a loop, which is impossible as M is irredundant.

On the other hand, suppose that $\tau_2\tau_3^\omega$ skips a loop. Now, (s_i, t) is reachable in $X \times B$ and so we have $t \in E_i$. By construction, $t \in E_j$ as well and so there is a path τ_1'' in $(X \to Y) \times B$ that ends in (s_j, t). But then $\tau_1''\tau_2\tau_3^\omega$ is an accepting path in M that skips a loop, which is again impossible as M is irredundant. □

3.3.4 Small-Model Property: Noose-Heavy Case

We now consider the case of irredundant ω-loop models with loops concentrated in the noose. In this subsection, we show that given an irredundant ω-loop model with more than $K_\omega/2$ loops in the noose N it is possible to split the noose $N = X \to Y \to Z$ so that pumping the noose $X \to Y^k \to Z$ creates irredundant ω-loop models with arbitrary many loops.

Throughout this subsection, let M be an irredundant ω-loop model with at least $K_\omega/2 = 2^{4|B|^2}$ loop in the noose.

Denote the state where the stem and the noose meet by s. Then split the states into the following four (possibly overlapping) categories:

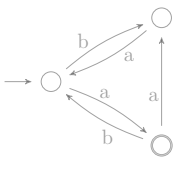
$$
\begin{aligned}
E_i^p &:= \{b \in B \mid \Pi_{(s,p)}^{(s_i,b)} \neq \emptyset\} \\
F_i^p &:= \{b \in B \mid \exists \pi \in \Pi_{(s,p)}^{(s_i,b)} \,.\, \pi \text{ visits a final state}\} \\
G_i^q &:= \{b \in B \mid \Pi_{(s_i,b)}^{(s,q)} \neq \emptyset\} \\
H_i^q &:= \{b \in B \mid \exists \pi \in \Pi_{(s_i,b)}^{(s,q)} \,.\, \pi \text{ visits a final state}\}
\end{aligned}
$$

If $n > 2^{4|B|^2} = K_\omega/2$ then by the pigeonhole principle, there are indices $i < j$ such that

$$(E_i^p, F_i^p, G_i^q, H_i^q) = (E_j^p, F_j^p, G_j^q, H_j^q) \text{ for every } p, q \in B$$

then the segment of the noose between s_i and s_j can be pumped.

Precisely, write the model M as $P \to N$ where P is the stem and N is the noose. Split (see Figure 3.13) the noose N at states s_i and s_j and write N as $N := X \to$



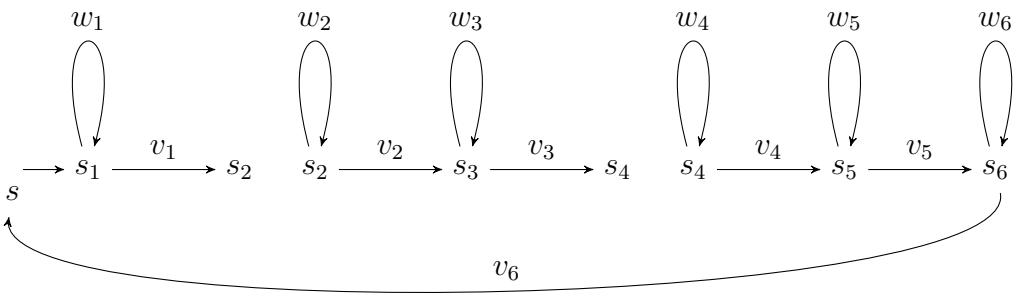


Figure 3.13: Splitting of the noose of an irredundant ω-loop model (noose-heavy case). The noose is split at states s_2 and s_4.

$Y \to Z$ where each X, Y and Z are loop models and the first state of X is identified with the last state of Z. Then the Y part can be pumped:

Lemma 3.3.13. *Let $M = (n, \overrightarrow{w}, \overrightarrow{v}, s)$ be an irredundant ω-loop model. Write $M = P \to N$ where $N = X \to Y \to Z$ is the noose. Then $M' = P \to N'$ is also a irredundant ω-loop model where $N' = X \to Y \to Y \to Z$ is the noose of M'.*

Proof. Suppose, to the contrary, that there is a lasso-shaped accepting path $\pi = \pi_1 \pi_2^\omega$ in $M' \times B$ such that π skips a loop in M'. Denote the skipped loop by w_t. If π is completely contained in $P \times B$ then π is an accepting run in $M \times B$, which is impossible.

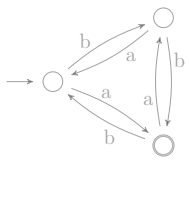
By unwinding π_2 if necessary, we can assume that π_2 starts in the state s of M'.

Suppose that π traverses the noose infinitely many times. In other words, suppose that it does not stay forever in some loop of the noose. Then we can factor π_2 into individual traversals of the noose and write π_2 as $\pi_2 = \alpha_1 \dots \alpha_k$ where each α_i starts and ends in the state s of M' and traverses the noose exactly once (s does not appear inside α_i). By construction, none of the α_i's traverses the loop w_t.

We shall create a path ρ in the noose N that skips the equivalent of w_t but visits a final state infinitely often. The path ρ is obtained by concatenating $\beta_1 \dots \beta_k$ where each β_i is a modification of α_i. We shall construct each β_i in such a way that it starts and ends in the same state as α_i does.

Fix $1 \leq i \leq k$ and suppose that $(s, p), (s, q)$ are the first and the last state of α_i respectively. Let s_a, s_b, s_c be the first state of the first copy of Y, the second copy of Y and Z respectively. Let $f(j)$ be the index of the first occurrence of the state s_j in α_i. We consider two cases depending on the position of w_t in N'.

If $t \geq b$ then denote the suffix of α_i starting at index $f(b)$ by $\tau_2 = \alpha_i[f(b) \dots)$. Observe that τ_2 is a path in $(Y \to Z) \times B$ that skips the loop w_t. Suppose that



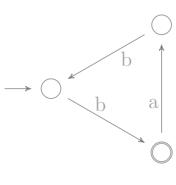
$\alpha_i(f(b)) = (s_b, g)$ for some state g of B. Then the state g is reachable from p in B and so $g \in E_b^p$. By construction, $g \in E_b^p = E_a^p$. Hence, there is a path τ_1 from (s, p) to (s_a, g). If, furthermore, the path $\alpha_i[1 \dots f(b)]$ visits a final state then $g \in F_b^p = F_a^p$ and so we can take τ_1 to be a path from (s, p) to (s_a, g) that visits a final state. Now, τ_1 is a path in $X \times B$. Hence we take $\beta_i := \tau_1\tau_2$. Note that β_i skips the equivalent of w_t and that β_i visits a final state if and only if α_i does and $\text{first}(\alpha_i) = \text{first}(\beta_i), \text{last}(\alpha_i) = \text{last}(\beta_i)$.

Now consider the case $t < b$ and denote the prefix of α_i ending at index $f(b)$ by $\tau_1 = \alpha_i[1 \dots f(b)]$. Observe that τ_1 is a path in $(X \to Y) \times B$ that skips the loop w_t. Suppose that $\alpha_i(f(b)) = (s_b, g)$ for some state g of B. Then the state q is reachable from g in B and so $g \in G_b^q$. By construction, $g \in G_b^q = G_c^q$. Hence, there is a path τ_2 from (s_c, g) to (s, q). If, furthermore, the path $\alpha_i[f(b)) \dots)$ visits a final state then $g \in H_b^q = H_c^q$ and so we can take τ_2 to be a path from (s_c, g) to (s, q) that visits a final state. Now, τ_2 is a path in $Z \times B$. Hence we take $\beta_i := \tau_1\tau_2$. Note that β_i skips w_t and that β_i visits a final state if and only if α_i does and $\text{first}(\alpha_i) = \text{first}(\beta_i), \text{last}(\alpha_i) = \text{last}(\beta_i)$.

In either case, the obtained path $\rho = \beta_1 \dots \beta_k$ skips the equivalent of w_t in N, visits a final state and starts in the same state as π_2 does. The path π_1 may traverse the noose a finite number of times. In exactly the same way as above, we can then transform π_1 into a path π_1' in M so that the path $\pi_1'\rho$ is an accepting path in $M \times B$. But this is impossible as M is an irredundant ω-loop model.

This settles the case when π visits the noose infinitely often. If π visits the noose only finitely many times then let π_1' be as in the previous paragraph. Then $\pi_1'\pi_2$ is a lasso shaped path that stays forever in the noose N.

Finally, it remains to be shown that there is an accepting path in $M' \times B$. The proof is analogous to the argument above. Let $\pi = \pi_1\pi_2^\omega$ be an accepting lasso-shaped path in $M \times B$. Then factor π_2 into individual traversals of the noose: $\pi_2 = \alpha_1 \dots \alpha_k$. Consider some α_i and factor it as $\alpha_i = \tau_1\tau_2$ where τ_1 is the prefix up to and including the first occurrence of s_i. So τ_1 is a path in $X \times B$ and τ_2 is a path in $(X \to Y) \times B$. Then $\text{last}(\tau_1) = (s_i, b)$ for some state $b \in B$. So $b \in E_i^p$. Hence, $b \in E_j^p$ as well and so there is a path τ_1' from (s, p) to (s_j, b). If, in addition, τ_1 visits a final state, then $b \in F_i^p = F_j^p$ and so we can then assume that τ_1' visits a final state as well. So set $\beta_i = \tau_1'\tau_2$. Then β_i is a path in $(X \to Y \to Y \to Z) \times B$ that visits a final state if and only if α_i does. Replacing each α_i by β_i we obtain an accepting path in $M' \times B$ as required. □



As in the previous section, the above proof easily generalises to pumping Y arbitrary many times:

Lemma 3.3.14. *Let $M = (n, \overrightarrow{w}, \overrightarrow{v}, s)$ be a irredundant ω-loop model. Write $M = P \to N$ where $N = X \to Y \to Z$ is the noose. Then for any $k \in \mathbb{N}$ we have $M' = P \to N'_k$ is also a irredundant ω-loop model where $N'_k = X \to Y^k \to Z$ is the noose of M'.*

As a corollary, we have the equivalence $(b) \iff (c)$ from Theorem 3.3.6 that it suffices to consider irredundant ω-loop models between K_ω and $2K_\omega$ loops.

Lemma 3.3.15. *For every $k \in \mathbb{N}$ there is an irredundant ω-loop model with at least k loops if and only if there is an irredundant ω-loop model with n loops where $2K_\omega \geq n > K_\omega$.*

3.3.5 From General to Irredundant ω-Loop Models

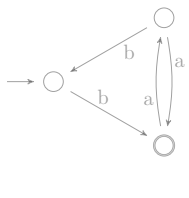
In the previous sections we have proved all but one implication in Theorem 3.3.6 relating nonlinear Büchi automata and irredundant ω-loop models. We now prove the remaining implication, $(a) \implies (b)$:

"Büchi automaton B does not have a linear completeness thresholds"
$\implies$
"For every $k \in \mathbb{N}$ there exists an irredundant ω-loop model with at least k loops"

As in the finite case we extract arbitrary large irredundant ω-loop models from general models witnessing nonlinearity of B. Given a general model M, recall that in case of regular languages we used normalisations (Definition 3.2.26) to show that nonlinearity of an automaton is always witnessed by irredundant loop models. A normalisation N is a loop model that embeds into M and the embedding is supplied by a path in M so that loops and arcs of N correspond to locally minimal paths in M. We now give a similar argument for ω-regular languages. We define an analogous notion of ω-normalisation P, which is an ω-loop model that embeds into M.

Definition 3.3.16. *An ω-normalisation $N_\omega = (N, N')$ with n loop bundles consists of two normalisations N and N' with m and m' loop bundles respectively. The normalisations satisfy $\text{last}(\alpha_m) = \text{first}(\alpha'_0) = \text{last}(\alpha'_m)$ and $n = m + m'$. The ω-normalisation N_ω is obtained by identifying the last state of N with the first and the last state of N'.*

Loops and arcs of N_ω consistently map into locally minimal loops and paths in M and so:



Lemma 3.3.17. *For a normalisation N_ω, it holds that $L(N_\omega) \subseteq L(M)$.*

Proof. As in the proof of Lemma 3.2.27 on page 49, observe that each path through N naturally corresponds to a path through M. □

If $\pi = \pi_1\pi_2^\omega$ is a lasso-shaped path then the embedding of the stem N and the noose N' arises from π_1 and π_2, respectively. By instantiating the following result independently for π_1 and π_2 and gluing the normalisations correspondingly, we obtain an accepting ω-normalisation N_ω.

Theorem 3.3.18. *Let states $m_1, m_2 \in M$ and $w \in \Sigma^*$ be such that $m_1 \xrightarrow{w} m_2$. Then there exists a normalisation N such that $|N| \leq |w|, \text{first}(\alpha_0) = m_1, \text{last}(\alpha_n) = m_2$ and there is a path π from the initial state of N to the last state of N such that $\text{word}(\pi) \sim w$, where the α_i's refer to the locally minimal paths from Definition 3.2.26.*

Proof. Exactly as in the case of regular languages (Theorem 3.2.30), however note that the definition of $\sim$ is different for ω-regular languages. □

As in Lemma 3.2.28, for an accepting ω-normalisation N_ω with k loop bundles it holds that $\text{sap}(M \times B) \leq \text{sap}(N_\omega \times B)$ the latter of which is bounded by a function of the number of loop bundles in N_ω. Thus, ω-normalisations provide a bound on $\text{sap}(M \times B)$ in terms of $\text{rd}(M)$.

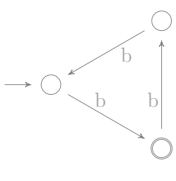
Lemma 3.3.19. *Let $N_\omega = (N, N')$ be an ω-normalisation with n loops bundles. Then $\text{sap}(M \times B) \leq 4n|B|^2C^2\,\text{rd}(M)$.*

Proof. The noose N' is visited at most $|B|$ times by the shortest accepting path in $N_\omega \times B$. Thus, every arc is taken at most $|B|$ times. Every loop, of which there are at most $n|B|$ many, is traversed at most $|B|$ times in total by the shortest accepting path in $N_\omega \times B$. Using Lemma 3.2.24 to bound the lengths of α's and β's, we obtain

$$\begin{aligned}
\text{sap}(M \times B) &\leq \text{sap}(N \times B) \\
&\leq (n+1)|B|C^2\,\text{rd}(M) + n|B|^2C^2(\text{rd}(M)+1) \\
&\leq 4n|B|^2C^2\,\text{rd}(M)
\end{aligned}$$

□

So if there is $k \in \mathbb{N}$ such that every model has an ω-normalisation with at most k loops then B is linear. Therefore, as in the Theorem 3.2.29, if a Büchi automaton is nonlinear then for every $k \in \mathbb{N}$ there must exist a model such that every accepting ω-normalisation has at least k loops.



Theorem 3.3.20. *If B is nonlinear then for every $k \in \mathbb{N}$ there is a model M_k such that $L(M_k \times B) \neq \emptyset$ and every ω-normalisation of M_k has at least k loop bundles.*

Then, analogously to the Theorem 3.2.32, the ω-normalisation with the smallest noose, and smallest stem in case of a tie, gives rise to an irredundant ω-loop model.

Theorem 3.3.21. *Let B be nonlinear. Then for every $k \in \mathbb{N}$ there exists a irredundant ω-loop model with at least k loops. Moreover, the irredundant ω-loop model has the property that every accepting lasso-shaped path traverses the noose infinitely many times.*

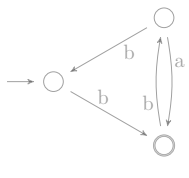
Proof. Let M_k be as defined above and let $N_\omega = (N, N')$ be the ω-normalisation of M with the smallest N' (least number of edges). In case of a tie, pick the ω-normalisation with the smallest N (least number of edges). Let $\pi = \pi_1 \pi_2^\omega$ be an accepting lasso-shaped path in $N_\omega \times B$. We shall prove that π_1 traverses every loop of N and π_2 traverses every loop of N'.

Suppose, to the contrary, that π_1 skips some loop $w_{i,j}$ of N. Let N_f be the normalisation obtained from N by deleting $w_{i,j}$ and the corresponding path $\beta_{i,j}$ and let N'_ω be the ω-normalisation obtained from N_f and N'. Note that N_ω and N'_ω have the same noose.

If the size k_i of the loop bundle containing the deleted loop $w_{i,j}$ is more than 1 ($k_i > 1$) then N_f is a normalisation smaller than N. But this is impossible.

If $k_i = 1$ then we eliminated the i-th loop bundle from N thereby concatenating v_{i-1} with v_i and α_{i-1} with α_i. The concatenated path might not be locally minimal in M, however, we can replace it by a shorter locally minimal path. Let N_l be the normalisation obtained by applying Theorem 3.2.30 to $\text{first}(\alpha_{i-1}), \text{last}(\alpha_i)$ and $v_{i-1}v_i$. Finally, let N'_f be the normalisation obtained by replacing $v_{i-1}v_i$ and $\alpha_{i-1}\alpha_i$ in N_f by N_l. Then N'_f is a smaller normalisation than N and (N_l, N') gives rise to a ω-normalisation smaller than N_ω. But this is again impossible. So π_1 takes every loop in N.

By a similar argument, we show that π_2 takes every loop in N'. Let $m = \text{last}(\alpha_m) = \text{first}(\alpha'_0)$ be the first state of the noose. By unwinding if necessary, we can assume that π_2 starts in (m, b) for some $b \in B$. Let N_f be the normal obtained by applying Theorem 3.2.30 to $\text{first}(\alpha_0), m$ and $\text{word}(\pi_1)$. Suppose, to the contrary, that π_2 skips some loop $w'_{i,j}$ of N'. Let N_s be the normalisation obtained from N' by deleting $w'_{i,j}$ and the corresponding path $\beta'_{i,j}$ and let $N'_\omega = (N_f, N_s)$ be the ω-normalisation obtained from N_f and N_s. Note that that the noose of N_ω is longer than the noose of N'_ω.



If $k'_i > 1$ then N_s is a normalisation smaller than N. But this is impossible. If $k'_i = 1$ then we eliminated the ith loop bundle from N' thereby concatenating v'_{i-1} with v'_i and α'_{i-1} with α'_i. The concatenated path might be not locally minimal in M, but again, we can replace it by a shorter locally minimal path. Let N_l be the normalisation obtained by applying Theorem 3.2.30 to $\text{first}(\alpha_{i-1}), \text{last}(\alpha_i)$ and $v_{i-1}v_i$. Finally, let N'_s be the normalisation obtained by replacing $v_{i-1}v_i$ and $\alpha_{i-1}\alpha_i$ in N_s by N_l. Then N'_s is a smaller normalisation than N' and (N_f, N'_s) gives rise to a ω-normalisation smaller than N_ω. But this is again impossible. So π_2 takes every loop in N'.

Now, we show that it is possible to transform N_ω so that every loops bundle is of size 1. During the transformation either the stem or the noose of N_ω gets flattened into a single path and hence the number of loops in the resulting irredundant ω-loop model is only at least $\frac{k}{2}$.

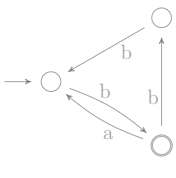
Fix an accepting lasso shaped path π through $N_\omega \times B$. Write π as $\pi = \pi_1\pi_2^\omega$. There are two cases depending on whether the loops of N_ω are concentrated in N or in N'. If m - number of loop bundles in N - is greater than $k/2$ then use the following irredundant ω-loop model $N_t = (N_{fin}, \text{word}(\pi_2))$. The expression N_{fin} denotes the normalisation as obtained for regular case by applying the argument from Theorem 3.2.32 to N. That is, we turn the prefix N into a ω-loop model and we fully unwind N' into a single path. Since every path through N_t can be transformed into a path through N_ω in a natural way, every accepting path through $N_t \times B$ takes every loop at least once.

If m' - number of loop bundles in N' - is greater than $k/2$ then write π_2 as $\pi_2 = (\tau_1 \dots \tau_l)^\omega$ where τ_i's are the individual traversals of the noose. The transformation consists of fours main stages, all of which are depicted in Figure 3.14. First, we completely unwind N into a single arc N_n spelling $\text{word}(\pi_1)$. To modify N', we first unwind the noose N' exactly l times thereby obtaining new model $N'_m = (\text{word}(\pi_1), N'_l)$.

Now, for every $1 \leq i \leq l$ and every loop bundle $1 \leq j \leq m'$ let $w_{i,j}$ be the word spelled by τ_i while traversing the jth loop bundle of N'. Finally, we replace the jth loop bundle in the ith copy of N' in N'_m by $w_{i,j}$. Denote the obtained model by N_f.

By the construction, every accepting lasso shaped path in $N_f \times B$ takes at least $m' \geq \frac{k}{2}$ loops.[5]

[5]Every path through N_f naturally embeds into a path through N' such that a traversal of a loop in N_f corresponds to a traversal of a loop bundle in N'. Since every accepting path in $N' \times B$ visits every loop bundle, every accepting path in $N_f \times B$ visits at least $m' \geq \frac{k}{2}$ loops



Now, the model N_f might not be irredundant. By taking minimal subset of loops we show that N_f can be made irredundant. For every accepting path π through $N_f \times B$ let L_π be the set of all loops of N_f traversed by π. By the paragraph above, $|L_\pi| \geq k/2$. Let L_m be a minimal set of L_π's with respect to set inclusion.

Finally, let N_m be obtained from N_f by keeping only the loops in L_m. By construction, there is an accepting path in $N_m \times B$ and every accepting path through $N_m \times B$ traverses every loop. Hence, N_m is irredundant. □

3.3.6 PSPACE Decision Procedure for ω-Regular Languages

In the previous subsections we characterised nonlinear Büchi automata in terms of irredundant ω-loop models. In this section we use the characterisation and present a decision procedure that given a Büchi automaton B determines whether B has linear completeness threshold.

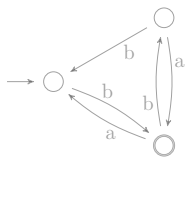
The decision procedure works by guessing a large enough irredundant ω-loop model M. Recall that in order to obtain irredundant ω-loop models with arbitrary many loops we took a model with at least K_ω loops and depending on the concentration of loops, we either pumped the stem or the noose. The decision procedure begins by nondeterministically guessing which of the two cases holds.

In the case of large stem, we assume, by unwinding the noose if necessary, that the noose is a simple loop w without any nested loops. Then we reuse with minor modification the algorithm for regular languages to guess large enough stem and hence irredundant ω-loop model. In more detail, recall from page 39 that for each i we use the sets E_i and F_i to denote the reachable states of B and the states of B reachable by a path skipping a loop in the prefix consisting of first i loops and arcs.

The decision procedure checks that

- there is no accepting path in $M \times B$ that stays in the stem forever. This is achieved by checking that $\forall b \in E_i \,.\, w_i^\omega \notin L_b(B)$ for every index i.
- There is no accepting path that skips a loop in the stem. This can be achieved by checking that $\forall b \in F_n \,.\, w^\omega \notin L_b(B)$.
- There is an accepting path in $M \times B$. This can be achieved by checking $\exists b \in E_n \,.\, w^\omega \in L_b(B)$.

The notation $L_b(B)$ denotes the set of words accepted by B starting in b instead of in the initial state q_0. Formally, $L_b(B) = \{w \in \Sigma^* \mid \exists \pi.\, \mathrm{word}(\pi) = w \ \wedge\ \mathrm{first}(\pi) =$



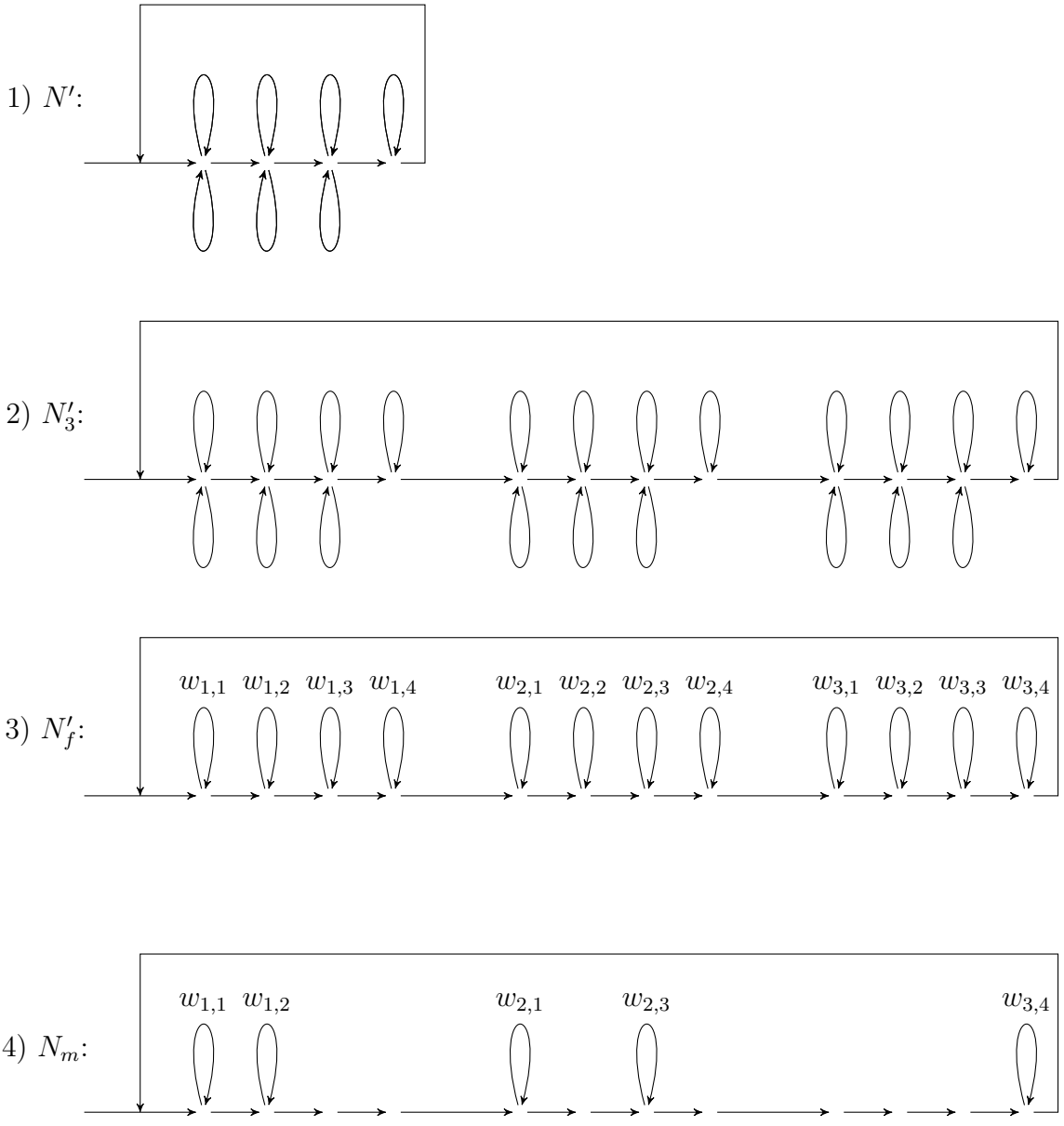
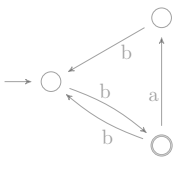


Figure 3.14: Example of turning N' into M_m.



$b \wedge \text{last}(\pi) \in F\}$. Note that the check $w^\omega \in L_b(B)$ can be easily implemented in PSPACE as it suffices to restrict to first $|B|$ iterations of w.

3.3.7 PSPACE Decision Procedure for ω-Regular Languages. Noose-Heavy Case

In the previous section we showed how to guess an irredundant ω-loop model with at least K_ω loops in the noose. In this section we show how to guess an irredundant ω-loop model M with the number of loops between K_ω and $2K_\omega$. Recall (Theorem 3.3.6) that this suffices to show that B is not linear.

A possibility is to guess an ω-loop model of such a size completely and then check that it is irredundant. This would yield an EXPSPACE algorithm. In general, we cannot guess the model on-the-fly as an accepting path may traverse the noose of the model more than once and so we have to keep track of the loops taken in each traversal of the noose.

We now state a slightly different condition that is sufficient and necessary to show that B is nonlinear. Moreover, the condition can be checked in PSPACE.

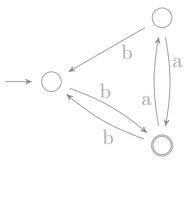
First, by unwinding the stem if necessary, we assume that the stem does not contain any loops. Therefore, the algorithm only checks the existence of a large noose. Second, for a lasso-shaped path $\pi = \pi_1\pi_2^\omega$, we can write $\pi_2 = \tau_1 \dots \tau_k$ where τ_i is the ith traversal of the noose by π. That is, $\text{first}(\tau_i) = \text{last}(\tau_i) = s$ and there are no other occurrences of s in τ_i. Let $\text{taken}(\tau_i)$ be the set of all loops taken by τ_i. We define $\text{minTaken}(\pi) = \min_i |\,\text{taken}(\tau_i)|$ to be the minimum number of loops visited during some single traversal of the noose. We can now state the sufficient and necessary condition:

Lemma 3.3.22. *The following are equivalent:*

(a) There is an irredundant ω-loop model M with at least K_ω loops in the noose.

(b) There is an ω-loop model M with at most $2|B|K_\omega$ loops such that for every accepting path π in $M \times B$ we have $\text{minTaken}\,\pi \geq K_\omega$.

Proof. Suppose that (a) holds. Then, by Lemma 3.3.14, there is an irredundant ω-loop model M' with the number of loops between $|B|K_\omega$ and $2|B|K_\omega$. Let π be an accepting path in M'. By considering a shortest path, we can assume that π traverses the noose at most $|B|$ times. But as π takes every loop, at least $|B|K_\omega/B = K_\omega$ loops are traversed in some single traversal of the noose.



Conversely, let M be an ω-loop model satisfying condition (b). For a path π let L_π be the set of loops traversed by π. By the assumption, for every accepting path π in $M \times B$ we have $|L_\pi| \geq K_\omega$. Let L_m be a minimal, with respect to set inclusion, set L_π for some accepting path π. Then remove from M all loops not in L_m. This creates an irredundant ω-loop model with at least K_ω loops in the noose. □

Notice that the condition (a) is sufficient and necessary to witness nonlinearity of B (Theorem 3.3.6). The decision procedure guesses loop-by-loop the noose containing between K_ω and $2|B|K_\omega$ loops and checks that it satisfies the condition (b) as follows.

Denote the state where the stem and the noose meet by s. Then for each pair of states $p, q \in B$ the algorithm iteratively calculates (Algorithm 2) the numbers A_{pq} and B_{pq} denoting the minimum number of loops on some single traversal of the noose from state (s, p) to state (s, q) with and without visiting a final state, respectively.

The algorithm then uses these values to check whether for some reachable state $t \in B$ there is a path from (s, t) to (s, t) traversing the noose (possibly several times) and visiting a final state so that during every single traversal of the noose the path visits less than K_ω loops. If this is the case then the noose does not satisfy the condition (b) from Lemma 3.3.22 and hence the noose does not witness nonlinearity of B.

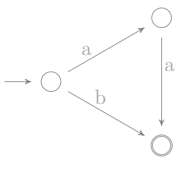
We now describe how to check that every accepting path visits at least K_ω loops in some traversal of the noose. Consider the following graph G. For every state $b \in B$ graph G contains vertices b_+ and b_-. Intuitively, the vertex b_- correspond to a path finishing in the state b of B. The vertex b_+ corresponds to a path finishing in s and visiting a final state. Then, for every $p, q \in B$ such that $B_{pq} < K_\omega$ we add the edges (p_+, q_+) and (p_-, q_-) to G. Further, for every $p, q \in B$ such that $A_{pq} < K_\omega$ we add the edges (p_+, q_+) and (p_-, q_+) to G.

Let $t \in B$ be some state of B such that (s, t) is reachable by a path from the initial state of M. If there is a path from t_- to t_+ in G then there is an accepting path such that every single traversal of the noose visits less than K_ω loops.

If there is no such path then every accepting path through the noose traverses at least K_ω loops and the condition (b) from Lemma 3.3.22 is satisfied.

Then the decision procedure checks whether an accepting path exists. By building a similar graph (without the restriction $A_{pq}, B_{pq} < K_\omega$), we can establish the existence of an accepting path.

Finally, the decision procedure ensures that it is impossible to remain stuck inside a single loop in the noose forever and accept. This check is done on-the-fly as follows. The line marked by (*) in Algorithm 2 is replaced by the code shown in Algorithm 3.



Algorithm 2 Algorithm for calculating the values A_{pq} and B_{pq}. The noose is built by guessing w_i and v_i one at a time. The values are updated based on whether a path exists from p to q that takes w_i and visits a final state.

for $p, q \in B$ **do**

$$A_{pq} \leftarrow \begin{cases} 0 & \text{if } p = q \text{ and } p \in F \text{ is final} \\ -\infty & \text{otherwise} \end{cases}$$

$$B_{pq} \leftarrow \begin{cases} 0 & \text{if } p = q \\ -\infty & \text{otherwise} \end{cases}$$

end for

for $i = 1$ **to** K **do**

$T_w, R_g \leftarrow$ initialise appropriately by guessing w letter by letter

$(*)$

$$A'_{pq} = \min \left(\begin{array}{ll} \{A_{pr} + 1 \mid (r,q) \in R_{w^k v} \text{ where } 1 \leq k \leq |B|\} & \cup \\ \{B_{pr} + 1 \mid (r,q) \in T_{w^k v} \text{ where } 1 \leq k \leq |B|\} & \cup \\ \{A_{pr} \mid (r,q) \in R_v\} & \cup \\ \{B_{pr} \mid (r,q) \in T_v\} & \end{array} \right)$$

$$B'_{pq} = \min \left(\begin{array}{ll} \{B_{pr} + 1 \mid (r,q) \in R_{w^k v} \text{ where } 1 \leq k \leq |B|\} & \cup \\ \{B_{pr} \mid (r,q) \in R_v\} & \cup \end{array} \right)$$

$A_{pq} = A'_{pq}$

$B_{pq} = B'_{pq}$

end for

The added code iteratively updates the set of states $BadStart \subseteq B$ such that if the noose is entered in those states of B then an accepting path exists that remains forever stuck in the current loop. So the decision procedure checks that the set of states of B reachable after the stem and $BadStates$ are disjoint.

Algorithm 3 Code snippet for calculating the set of states $BadStart \subseteq B$ such that if the noose is entered when the state of B is in $BadStart$ then there is an accepting path staying inside some loop.

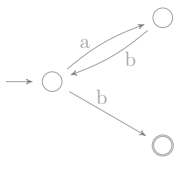
$Bad = \{t \mid (t,t) \in T_{w^k} \text{ for some } 1 \leq k \leq |B|\}$

$BadStart = BadStart \cup \{p \mid B_{pb} > 0 \wedge b \in Bad\}$

Since K_ω is singly exponential in $|B|$ and we can assume that the noose and every loop is taken at most $|B|$ times, only polynomially many in $|B|$ bits are needed to store A_{pq}, B_{pq} and all intermediate values. Hence, the algorithm can be implemented in PSPACE.

Finally, observe that by taking C to be a nonlinear Büchi automaton in (Theorem 3.2.33) we obtain the corresponding hardness result.

Theorem 3.3.23. *It is* PSPACE*-hard to determine whether a Büchi automaton B has a linear completeness threshold.*



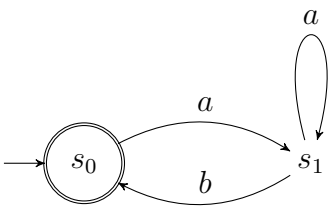


Figure 3.15: A quadratic Büchi automaton.

3.3.8 Quadratic Completeness Threshold

Unlike in the case of regular languages, there are Büchi automata with the completeness threshold strictly between linear and exponential. In the theory of regular languages, we were able show (Theorem 3.2.19) that if an automaton is not linear then it has at least exponential completeness threshold. The construction (Figure 3.7), however, does not work for infinite strings. In the finite case, it is always true that a final state of B is visited only at the very end. However, in the case of infinite words, we may not rule out the possibility that an accepting path exists that is stuck inside some T_k forever. And so it is possible that there is an accepting path which loops forever in a submodel of the model from Figure 3.7. In fact, we now show that there is a simple Büchi automaton with precisely quadratic completeness threshold. On the other hand, there are still Büchi automata with exponential completeness threshold, similar to that depicted in Figure 3.10.

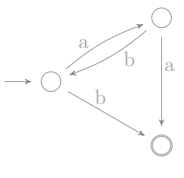
Theorem 3.3.24. *The automaton in Figure 3.15 is exactly quadratic.*

Proof. Family of irredundant ω-loop models witnessing at least quadratic completeness threshold has $w = a^*$ and $v = b$.

Let M be a model such that $L(M \times B) \neq \emptyset$ and let $\alpha\beta^\omega$ be the shortest accepting lasso-shaped path in $M \times B$. By proving results on the structure of loops of M traversed by α and β we show that the model induced by π is almost an irredundant ω-loop model.

Suppose that there are $i < j$ such that $\alpha(i) = (m, x)$ and $\alpha(j) = (m, y)$. Let p be the label of the edge from $\alpha(i)$ to $\alpha(i+1)$ and q be the label of the edge from $\alpha(j)$ to $\alpha(j+1)$ - if $\alpha(j)$ is the last state of α then take $\beta(1)$ instead. Since s_0 cannot be followed by b there are 9 possibilities for the tuple (x, y, p, q) (see Figure 3.16)

If $x = y$ then $\alpha[1 \ldots i]\alpha[j \ldots)\beta^\omega$ is a shorter lasso-shaped path. Thus, no such loop appears in α.



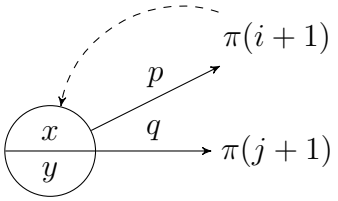


Figure 3.16: Structure of a loop in α.

If $q = a$ then $\alpha(j+1) = (n, s_1)$ for some $n \in M$ such that m is connected to n by an edge. But this state is reachable from $\alpha(i)$ and so the path $\alpha[1 \dots i]\alpha[j+1 \dots)\beta^\omega$ is a shorter accepting lasso shaped path. Thus, no such loop appears in α.

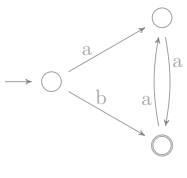
Finally, we are left with the situation where all loops satisfy $x = s_0, y = s_1, p = a$ and $q = b$. Now, $\alpha(i+1) = (n, s_1)$ which is a possible successor of $\alpha(j)$. So if there is some b between $\alpha(i)$ and $\alpha(j)$ then $\alpha[1 \dots i]\alpha[i+1 \dots j]^\omega$ is a shorter lasso-shaped accepting path. Hence, there cannot be any b between $\alpha(i)$ and $\alpha(j)$. That is, the word spelled by $\alpha[i \dots j]$ is a^{j-i} and since α is minimal, the projection of $\alpha[i \dots j]$ onto M is a simple loop. Therefore, there are no nested loops in the projection of α onto M and so the model induced by α is a loop model.

This concludes α. Now suppose that there are $i < j$ such that $\beta(i) = (m, x)$ and $\beta(j) = (m, y)$ and it is not the case that $i = 1$ and $j = |\beta|$. Let p be the label of the edge from $\beta(i)$ to $\beta(i+1)$ and q be the label of the edge from $\beta(j)$ to $\beta(j+1)$ - if $\beta(j)$ is the last state of β then we take $\beta(1)$ instead. As above, there are 9 possibilities for (x, y, p, q). Denote the segment $\alpha[i \dots j]$ by ρ.

If $x = y$ then either ρ visits a final state in which case $\alpha\beta[1 \dots i)\beta[i \dots j)^\omega$ is a shorter lasso shaped path, or ρ does not visit a final state in which case the path $\alpha(\beta[1 \dots i)\beta[j \dots))^\omega$ is a shorter lasso shaped accepting path. But this is not possible as π is the shortest accepting lasso shaped path. So $x \neq y$ and we are left with only four remaining cases. We shall consider every one of them.

If $x = s_0, y = s_1$ and $p = q = a$. Then $\beta(i)$ is a final state and $\beta(j+1) = (n, s_1)$ for some $n \in M$ such that m is connected to n by an edge. But this state is reachable from $\beta(i)$ and so the path $\alpha(\beta[1 \dots i]\beta[j+1 \dots))^\omega$ is a shorter accepting lasso shaped path.

If $x = s_1, y = s_0$ and $p = q = a$. Then $\beta(i+1) = (n, s_1)$ which is a possible successor state of $\beta(j)$. Hence, $\alpha\beta[1 \dots i]\beta[i+1 \dots j]^\omega$ is a shorter accepting lasso shaped path visiting the state $(m, s_0) = \beta(j)$ infinitely often.



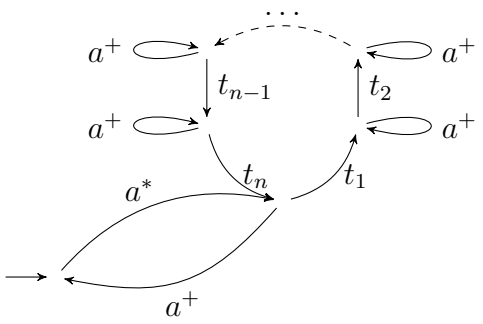


Figure 3.17: Possible shape of the noose of the model induced by the path π from Theorem 3.3.24. Each t_i begins and ends with b, $t_i \in b + b\Sigma^* b$.

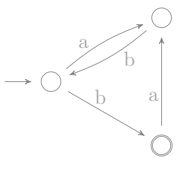
If $x = s_0, y = s_1, p = a$ and $q = b$ then as above, it is possible to show that word$(\beta[i \dots j]) = a^{j-i}$, the projection of $\beta[i \dots j]$ onto M is a simple loop. So if there are only loops of this form in β then the model induced by β is a loop model with the first and the last state identified – a simple noose.

Finally, suppose that there is a loop of the form $x = s_1, y = s_0, p = b$ and $q = a$. Observe that $\beta(j+1) = (n, s_1)$ for some $n \in M$ and so $\beta(j+1)$ is a valid successor of $\beta(i)$. If there is a b in word$(\beta[j \dots)\beta[1 \dots i])$ then we can skip the loop thereby obtaining a shorter lasso-shaped accepting path $\alpha(\beta[1 \dots i]\beta[j+1 \dots))^\omega$. So there is no b on that path, hence the word it spells consists only of a's and so the projection of the path $\beta[j \dots)\beta[1 \dots i]$ onto M is a simple loop. Suppose that there is another loop of the type $x = s_1, y = s_0, p = b, q = a$ at position $k \dots l$ where $i \leq k < l \leq j$ such that $(i, j) \neq (k, l)$. Now, we know that $\beta(i+1)$ and $\beta(j)$ are accepting. Since the loop $\beta[k \dots l]$ is properly nested in the loop $\beta[i \dots j]$ the removal of the loop $\beta[k \dots l]$ leaves at least one of these two states which gives rise to a shorter lasso-shaped accepting path. Therefore, there are no nested loops inside $\beta[i \dots j]$ of this form. Thus, all nested loops are as in the previous paragraph ($x = s_0, y = s_1, p = a, q = b$). It follows that the submodel of M induced by $\beta[i \dots j]$ is a loop model and the projection of β^ω onto M is as in Figure 3.17.

So, α and β correspond to traversals of (almost) loop models and the models of this shape are clearly at most quadratic. □

3.4 Discussion

The results presented in this chapter settle the main open questions listed in Section 6 of [KOS+11]: it is decidable, and in fact PSPACE-complete, whether a regular (resp. ω-regular) specification has a linear completeness threshold, provided the specification



is given as an automaton (resp. Büchi automaton). Moreover, two dichotomies are at play: a regular specification either has a linear or an exponential completeness threshold, whereas an ω-regular specification has the completeness threshold that is either linear or at least quadratic.

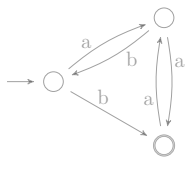
That is, for regular languages, the completeness thresholds is either asymptotically as good as possible or asymptotically as bad possible. There is no intermediate completeness threshold.

For ω-regular languages, analogously to regular languages (Theorem 3.2.21), it is easily seen that exponential completeness threshold is asymptotically the worst possible. However, the magnitude of completeness thresholds for ω-regular specifications between quadratic and exponential is more subtle. We conjecture that in the case of ω-regular specifications, there is in fact a *trichotomy*: completeness thresholds are either linear, precisely quadratic, or precisely exponential.

To show that the completeness threshold is at least quadratic, the strategy employed in this thesis is to construct for every $n \in \mathbb{N}$ an irredundant ω-loop model with n loops each of which is of length n. To show that the completeness threshold is at least cubic, a conceivable strategy is to construct for every $n \in \mathbb{N}$ an irredundant ω-loop model model with n outer loops each with n nested loops with each nested loop of length n. We believe that by extending the current techniques it is possible to show such a result and hence obtain that if the completeness thresholds is more than quadratic then it is at least cubic.

By introducing more layers of nested loops, we believe it possible to show that the completeness threshold can be arbitrary polynomial and never anything in between. For example, to show that the completeness threshold of a Büchi automaton is at least $\mathrm{rd}(n)^k$, one can build a family of models with n loops, each with n nested loops, each nested loop with another n nested loops and so on, stopping when $k-1$ layers of nestedness are reached.

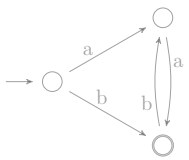
We, however, conjectured a stronger result. Precisely, we conjectured that for ω-regular languages if the completeness threshold is more than quadratic then it is exponential. The reason why quadratic completeness threshold occurs is that (as seen in Figure 3.17) it is possible for an accepting path to stay inside a loop with n nested loops. However, we believe that if the completeness thresholds is at least cubic then the above situation cannot occur as accepting paths are forced to visit nested loops in all n outer loops. In that situation we believe it be possible to create exponential family of models similar to the one shown in Figure 3.7 on page 44.



Another interesting question is the complexity of determining whether an ω-regular specification has a linear completeness threshold, *assuming the specification is provided as an* LTL *formula*. One immediately obtains an EXPSPACE upper bound through the translation of LTL formulas into (at most) exponentially-sized Büchi automata. We were able to show only PSPACE lower bound for LTL and, in fact, we conjecture that PSPACE suffices. Note also that the PSPACE-hardness result relies on the automaton under consideration being nondeterministic. We leave open the question of complexity of the decision procedure for *deterministic* automata.

If the automaton has linear completeness threshold then we would like to estimate the linearity constant as close as possible. We showed how to bound the constant, however, the bounds obtained are quite loose from both practical and theoretical point of view. The problem of *calculating and/or approximating* the constant more closely remains open.

Finally, model checking can be applied to structures in which either states or edges are labelled. We presented our work in the context of edge-labelled structures, in order to simplify our exposition; our main results also apply, by adapting the approach appropriately, to state-labelled structures (Kripke structures), although the careful verification of this fact is a somewhat laborious exercise.



Chapter 4

Path Checking of Temporal Logics

A central problem underlying BMC and model checking in general is to determine whether a given execution of an automaton satisfies given temporal formula.

That is, given a concrete execution[1] w and a formula φ (drawn from a fixed ambient logic), does w satisfy φ? Not only this is a fundamental problem for the ambient temporal logic, but the complexity of this problem also plays a key role in the design and analysis of offline monitoring and runtime verification procedures [FS04, MN04]. The path-checking problem also appears in testing [ABG+05] and in Monte-Carlo-based probabilistic verification [YS02].

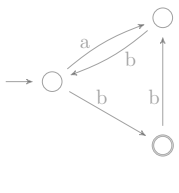
One can think of evaluating φ on w as as a special case of checking the inclusion $L(M) \subseteq L(B_\varphi)$ where M is a simple deterministic automaton with $L(M) = \{w\}$. The problem then becomes the so-called *path-checking* problem, sometimes known at the membership problem. In this chapter we study the path-checking problem for some of the most commonly used temporal logics (LTL, MTL and UTL).

Definition 4.0.1. *The* path-checking problem *for logic $\mathcal{L}$ is to determine, given a finite trace π and a formula φ of $\mathcal{L}$, whether $\pi, 1 \models \varphi$.*

The most-expressive temporal logic considered in this thesis is MTL. Let φ be an MTL formula and π a finite trace. Recall from Section 2.4 the recursive definition of $\pi, i \models \varphi$. The definition immediately gives rise to a dynamic-programming path-checking algorithm. For each subformula ψ of φ and every index i, the algorithm keeps track of whether $\pi, i \models \psi$. Starting from the smallest subformulae, the algorithm calculates the value for every ψ and i using the recursive definition. This yields a polynomial time algorithm checking whether $\pi, 1 \models \varphi$.

Theorem 4.0.2 ([MS03])**.** *The path-checking problem for* MTL *is in* P.

[1] In this chapter, all observations (paths, traces, words, etc.) considered are finite.



The above algorithm can be implemented in $O(|\pi||\varphi|)$ time and it follows that the path-checking problem for MTL and all its sublogics is solvable in polynomial time.

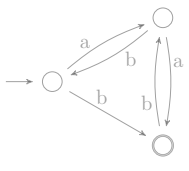
Although the path-checking problem is simply stated, determining the precise complexity of path-checking problems has proven to be quite challenging. The case of the most prominent temporal logic, LTL, was first investigated more than a decade ago [DS02, MS03], and at the time is was conjectured that the straightforward polynomial-time dynamic-programming algorithm is not optimal.[2] And indeed, using reductions to planar circuits and tree-contraction algorithms, it was recently proved [KF09] (summarised in Section 4.3) that LTL path checking allows an efficient parallel algorithm and lies in NC—in fact, in $\mathsf{AC}^1[\mathsf{logDCFL}] \subseteq \mathsf{AC}^2$. (This seminal result was rewarded by the ICALP 2009 best-paper award.)

In this chapter, we first present a partial converse to the above theorem, by showing that the evaluation of circuits drawn from a class of planar circuits studied in [LMS06] is reducible to LTL path checking. As a corollary, we conclude that any further progress in determining the precise complexity of the path-checking problem for LTL would therefore immediately entail more efficient evaluation algorithms than are known for this class of planar circuits. It is worth pointing out that augmenting this class of planar circuits with NOT gates makes the evaluation problem P-complete [Gol77]. It follows that the complexity of path checking is sensitive to non-monotone connectives, as allowing Boolean exclusive-or in formulae enables the evaluation of circuits from this augmented class, and is therefore itself P-complete.

Recently, the work [KF09] was extended [KF12] to a very restricted metric extension of LTL, in which only temporal operators of the form $\mathrm{U}_{\leq b}$ are allowed. In this chapter, we give a path-checking algorithm for full Metric Temporal Logic (MTL) with the same complexity—$\mathsf{AC}^1[\mathsf{logDCFL}]$—as the best presently known algorithm for LTL.

An examination of the algorithmic constructions for LTL path checking [KF09] and MTL path checking (Section 4.5) shows that the most intricate parts arise in handling the Until operator. We finish the chapter by showing that the removal of binary operators from the logic, yielding Unary Temporal Logic (UTL), leads to a much simpler path-checking problem, enabling us to devise an AC^1 tree-contraction algorithm for UTL path checking.

[2] The best known lower bound for LTL path checking is NC^1, which crudely arises from the NC^1-hardness of mere Boolean formula evaluation [BCGR92].



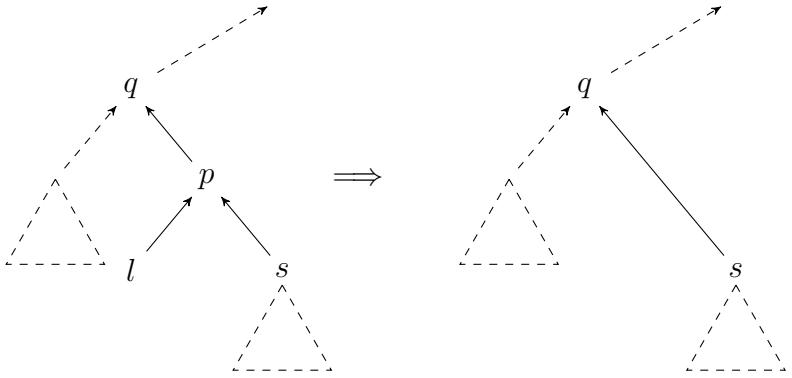


Figure 4.1: An example of a tree contraction step.

4.1 Tree Contraction

The path-checking algorithms devised in this chapter are based on the tree-contraction algorithm. Such an algorithm can be used to evaluate efficiently in parallel functions defined on a tree. In our settings we use the parse tree of the given formula φ. We revise tree-contraction algorithms. The presentation in this section follows that of [KR90].

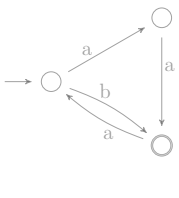
Let $T = (V, E)$ be a binary[3] tree, the tree contraction algorithm [KR90] reduces T to a single node using a sequence of tree contraction steps. Let $l \in T$ be a leaf, p be its parent, s its sibling and q its grandparent. A tree contraction step collapses the triple (l, p, s) into a single node. Formally, a new tree $T' = (V', E')$ is obtained from T as follows (see Figure 4.1): $V' = V \setminus \{l, p\}$

$$E' = \begin{cases} E \setminus \{(p,l),(p,s)\} & \text{if } p \text{ is the root of } T \\ (E \setminus \{(p,l),(p,s),(q,p)\}) \cup \{q,s\} & \text{otherwise} \end{cases}$$

Tree-contraction algorithms consist of iteratively applying the contraction step until a single node remains. Note that a contraction step is local and hence multiple non-interfering contractions can be performed in parallel. A tree contraction algorithm using only $\lceil \log n \rceil$ parallel steps exists [KR90] (Algorithm 4)

The algorithm can be implemented on EREW PRAM (exclusive read, exclusive write, random access memory machine) running in time $O(\log n)$ and total work $O(n)$ where n is the size of the tree. It is known [Vol99] that problems solvable by EREW PRAM algorithms in $O(\log n)$ parallel steps and polynomial total work are in AC^1.

[3] If a tree has nodes with only one child, we make the tree binary by introducing dummy children nodes.



Algorithm 4 Parallel tree-contraction algorithm requiring only $O(\log n)$ parallel steps. The algorithm appears in [KR90]

Number the leaves from left to right as $1, \ldots, n$
for $\lceil \log n \rceil$ iterations **do**
Apply tree contraction to all odd-numbered leaves that are left child of their parent.
Apply tree contraction to all odd-numbered leaves that are right child of their parent.
Shift out the rightmost bit in the labels of all remaining leaves.
end for

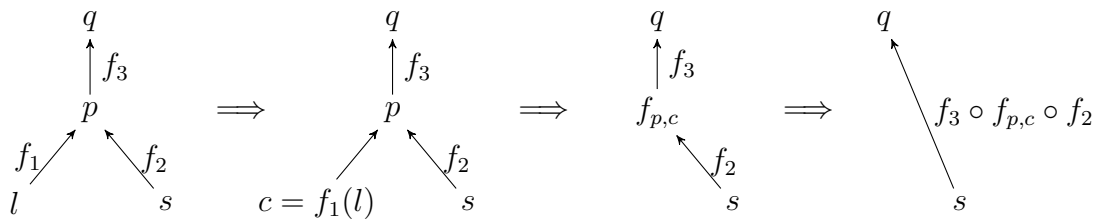


Figure 4.2: An example of a tree contraction step updating the functions associated with the edges.

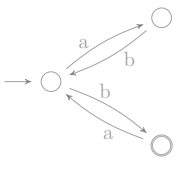
Tree-contraction algorithms can be used to evaluate a function f defined recursively on a tree. For example if φ is an LTL formula and π is a trace then the problem of evaluating $\pi \models \varphi$ can be phrased in such a way. The tree is the parse tree of φ, trace π supplies input values into leaves (atomic propositions) and every internal node is a function corresponding to the evaluation of the associated Boolean / temporal operator.

To evaluate a function on a tree using a tree-contraction algorithm, we extend the basic contraction step of a leaf l by partially evaluating the function associated with the parent p of l. Precisely, with each edge[4] $e = (u, v)$ we associate a function f_e with the interpretation that if the value of the subtree rooted at u is x then the operand supplied to v is $f_e(x)$. Then in each contraction step we update the functions appropriately. See Figure 4.2.

We need to update the function attached to the edge (s, q). The update consists of three steps (assuming p is not the root)

- Compute $c = f_1(l)$ (Recall that leaves are labelled with constants)
- Represent the function $\lambda x.p(c, x)$; denote it by $f_{p,c}$. (Use $f = \lambda x.p(x, c)$ if l is a right leaf)

[4]Edges are oriented towards the root



- Compute $g = f_3 \circ f_{p,c} \circ f_2$. Label (s, q) by g.

Note that all functions arising in the tree-contraction algorithm are obtained by composing the functions created in the second step above. If every (intermediate) function arising in the algorithm and every of the steps above can be implemented in complexity class C then the complexity of the entire tree-contraction algorithm is $\mathsf{AC}^1[C]$.

Note that when we representing the value of $f_{p,c}$ in the second step, the value of c is known and so $f_{p,c}$ is a function of only a single variable.

4.2 Conventions

In the rest of the chapter we use the following convention. The Boolean true and false are denoted by $\top$ and $\bot$, respectively and the set $\{\bot, \top\}$ is denoted by $\mathbb{B}$. Recall that a trace π of length n is a function assigning a truth value to every every atomic proposition at every index. We therefore identify $p \in \mathrm{AP}$ with an n-dimensional Boolean vector in $\mathbb{B}^n$ and write $p(i) = \pi(i, p)$.

The convention naturally extends to formulae. Given a trace π and formula φ, we represent the value of φ on π as the vector $v \in \mathbb{B}^n$ such that $v(i) = \top$ if and only if $\pi, i \models \varphi$.

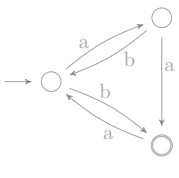
Now, we can think of LTL temporal operators as functions over n-dimensional Boolean vectors written in infix notation. For example, $\mathrm{U} : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^n$ is a function such that $(p \,\mathrm{U}\, q)(i) = \top$ if and only if there is $i \leq j \leq n$ such that $q(j) = \top$ and $p(k) = \top$ for all $i \leq k < j$. Similarly for other temporal operators.

Further, without loss of generality, we assume that all formulae consider in this chapter are in positive normal form, i.e., negation is applied only to atomic propositions.

4.2.1 Circuits Used

Recall that in Section 2.5 we defined the the standard circuit classes such as $\mathsf{NC}^1, \mathsf{AC}^1, \ldots$. In this chapter we use various less common classes of Boolean circuits, which we shall now introduce.

A circuit C is said to be *layered* if it can be partitioned into *layers* $C_0, \ldots, C_n$ such that each wire goes from C_i to C_{i+1} for some i. Thus, C_0 contains only input gates. In general, however, an intermediate layer of a layered circuit may also contain an input gate. A layered circuit is said to be *stratified* if all input gates appear in C_0.



A circuit is *upward planar* if there is a planar embedding such that every edge monotonically increases in the upward direction—the direction of the evaluation of C. A circuit is *upward layered (stratified)* if it is both upward planar and layered (stratified). Figure 4.5 on page 93 and Figure 4.6 on page 96 show upward stratified monotone circuits.

Now, gates in each layer C_i of an upward-layered circuit come with a natural left-to-right ordering as specified by the planar embedding (see e.g., Figure 4.5 on page 93). Therefore, each layer C_i of an upward-layered circuit consists of gates $\alpha_{i,j}$ in the left-to-right ordering. Since the circuit is planar, each gate $\alpha_{i,j}$ depends on a contiguous block of gates $\alpha_{i-1,l}, \ldots, \alpha_{i-1,r}$ a layer below. Furthermore, the wires do not cross: if $\alpha_{i,j}$ depends on $\alpha_{i-1,q}$ and $\alpha_{i,k}$ depends on $\alpha_{i-1,r}$ then $j \leq k \iff q \leq r$.

Further, it is known that the circuit-value problem (CVP) for upward-stratified circuits is P-complete [Gol77], CVP for monotone upward-stratified circuits is in $\mathsf{logDCFL}$ [CD06] and that CVP for monotone upward-layered circuits is in $\mathsf{AC}^1[\mathsf{logDCFL}]$ as was shown[5] in [LMS06].

We can now explain the LTL path-checking algorithm devised in [KF09]; the algorithm inspires our MTL and UTL path-checking algorithms presented later in the chapter.

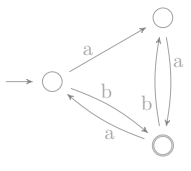
4.3 Tree-Contraction Algorithm for LTL Path Checking

We now briefly describe the tree-contraction algorithm devised in [KF09] for LTL path checking. The algorithm runs in $\mathsf{AC}^1[\mathsf{logDCFL}]$ and uses the parse of the given LTL formula as the underlying tree.

Let the input trace π be of finite length n. Then each leaf of the tree is labeled by a length-n vector denoting the truth values of propositions. The main observation of [KF09] is the efficient representation of the functions for temporal operators constructed in the second step above. It was shown in [KF09] that given $s \in \mathbb{B}^n$ how to represent for every temporal operator (such as Until, Since) the functions $\lambda x.p \,\mathrm{U}\, x$ using upward stratified circuits with n input and n output gates. Such circuits are called the (*transducer circuits*).

For example, consider the temporal Until operator. Then for a fixed $s \in \mathbb{B}^n$ the expression $s \,\mathrm{U}\, x$ is a function in x with domain and codomain equal to $\mathbb{B}^n$. Now,

[5]Recall that the notation $\mathsf{AC}^1[\mathsf{logDCFL}]$ denotes an AC^1 circuit with additional oracle gates calculating a function in $\mathsf{logDCFL}$.



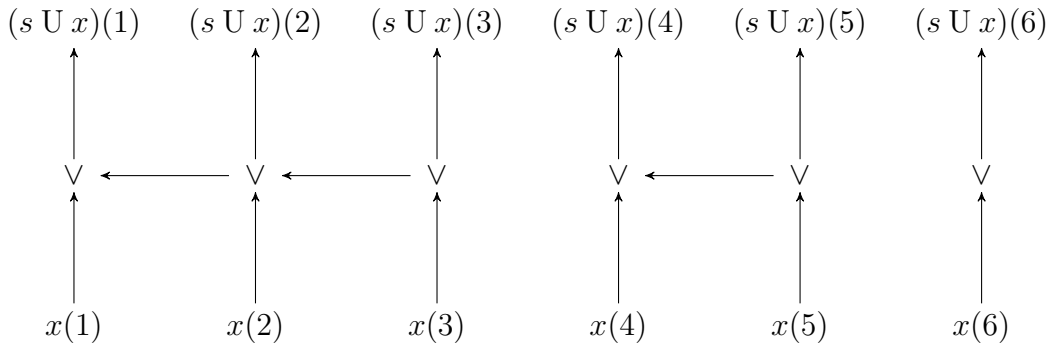


Figure 4.3: Circuit for $s \,\mathrm{U}\, x$ with $s = (\top, \top, \bot, \top, \bot, \bot)$.

Until operator satisfies the property $(s \,\mathrm{U}\, x)(i) = x(i) \vee (s(i) \wedge (s \,\mathrm{U}\, x)(i+1))$. Thus,

$$(s \,\mathrm{U}\, x)(i) = \begin{cases} x(i) & \text{if } s(i) = \bot \\ x(i) \vee (s \,\mathrm{U}\, x)(i+1) & \text{if } s(i) = \top \end{cases}$$

Such an expression gives immediately rise to a circuit C_s expressing the function $s \,\mathrm{U}\, x$ (see Figure 4.3).

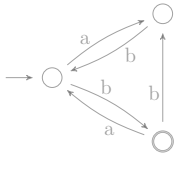
In general, consider a circuit C with n inputs and n outputs. By numbering the inputs and outputs appropriately, C defines a function from $\mathbb{B}^n$ to $\mathbb{B}^n$. In [KF09] it was shown how to construct circuits C such that for any $x \in \mathbb{B}^n$ we have $C(x) = s \,\mathrm{U}\, x$, $C'(x) = x \,\mathrm{U}\, s$, $C''(x) = s \,\mathrm{R}\, x$, $C'''(x) = x \,\mathrm{R}\, t$ as well as how to construct circuits for the past counterparts and Boolean operators.

Moreover, all the constructed circuits are upward stratified and monotone and it was shown in [KF09] that such circuits are closed under composition [KF09] and their evaluation and composition is in $\mathsf{logDCFL}$ [BLMS99]. This then immediately yields an $\mathsf{AC}^1[\mathsf{logDCFL}]$ tree-contraction algorithm.

Finally, suppose we extend LTL with some additional operator. Observe that as long as the partial evaluations of the operators are representable using upward-stratified monotone circuits, the path-checking problem for the extended logic is still solvable in $\mathsf{AC}^1[\mathsf{logDCFL}]$. In Section 4.5, we show how to construct corresponding transducer circuits for MTL temporal operators interpreted over a timed trace.

4.4 Reduction from Upward-Layered Circuit Value Problem to LTL Path Checking

In [KF09] (summarised in the previous section) an $\mathsf{AC}^1[\mathsf{logDCFL}]$ path-checking algorithm was given for LTL. The strategy introduced in [KF09] is to represent temporal



operators using a special class of planar monotone circuits together with a generic algorithm [CD06] as a subroutine to evaluate those circuits. Such circuits have a very special form (e.g., Figure 4.3), which led the authors of [KF09] to ask whether the complexity of the path-checking algorithm can be improved by devising specialised circuit-evaluation algorithms. In this section, we present evidence to the contrary, by showing that the evaluation of circuits drawn from a class of planar circuits studied in [LMS06] is reducible to LTL path checking; any further progress in determining the precise complexity of the latter would therefore immediately entail more efficient evaluation algorithms than are known for this class of planar circuits.

Recall upward layered monotone circuits from Section 4.2.1. Such a circuit is upward planar and consists of layers such that each layer depends only on the previous layers (e.g., Figure 4.5).

We show in this section that given an upward layered monotone circuit C with n gates and m wires it is possible to build an LTL formula φ over at most $2n$ propositions and a trace π of length $|\pi| \leq m$ such that C evaluates to $\top$ if and only if $\pi \models \varphi$.

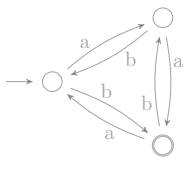
Denote the layers of C by $C_0, \ldots, C_k$ and the size of each C_i by n_i. Let $\alpha_{i,j}$ be the gates in C_i in the left-to-right order in the upward planar embedding of C. For each layer, we partition the trace into blocks—each of which stores the outputs of a gate in the layer. Figure 4.4 shows a valid partitioning. For example, in the figure, gate a occupies block $[1, 1]$, gate e occupies $[3, 5]$, gate g occupies $[1, 7]$, etc.

In general, a valid partitioning consists of a trace π and intervals $v(i, j)$ associated with each gate $\alpha_{i,j}$ such that $v(i, j)$ overlaps precisely with the blocks of the gates the gate $\alpha_{i,j}$ depends on. Formally,

Property 4.4.1.
- *intervals* $v(i, 1), v(i, 2), \ldots, v(i, n_i)$ *are disjoint and partition* $[1, |\pi|]$ *for every* i,
- *if* $\alpha_{i+1,j}$ *depends on* $\alpha_{i,p}, \alpha_{i,p+1}, \ldots, \alpha_{i,q}$ *then* $v(i + 1, j) \subseteq \cup_{r=p,\ldots,q} v(i, r)$ *and* $v(i + 1, j)$ *overlaps with each* $v(i, r)$ *for* $p \leq r \leq q$,

Suppose we are given a valid partitioning. Then for $i > 0$ and every $1 \leq j \leq n_i$ we build a formula context $\varphi_{i,j}$ mimicking the evaluation of the gate $\alpha_{i,j}$. Formally, when evaluated on appropriate proposition p, the formula context $\varphi_{i,j}(p)$ is constant on $v(i, j)$ and equals the output of $\alpha_{i,j}$ and $\varphi_{i,j}(p)$ is equal to p elsewhere.

We now show how to construct such formula context $\varphi_{i,j}$ corresponding to the gate e in Figure 4.4 assuming the gate is an OR gate. Denote the proposition that is true only in the interval $[i, j]$ and false otherwise is by $\chi_{i,j}$, i.e., $\chi_{i,j}(k) = \top$ if $i \leq k \leq j$ and $\chi_{i,j}(k) = \bot$ otherwise. Further suppose that the values of the blocks in



C_2 :	g	g	g	g	g	g	g
C_1 :	d	d	e	e	e	f	f
C_0 :	a	b	b	b	c	c	c

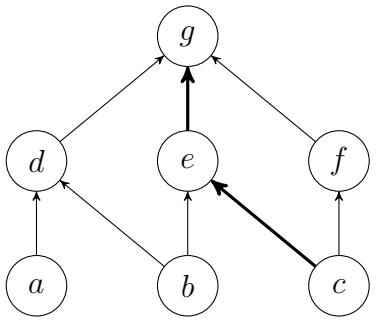


Figure 4.4: An upward layered circuit (on the right) with its partition (on the left). The path π for the gate labelled e is highlighted.

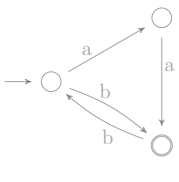
the first layer is $r = (a, b, b, b, c, c, c) \in \mathbb{B}^7$ for some $a, b, c \in \mathbb{B}$. Recall that $(\varphi \mathrm{U} \psi)(i) = \psi(i) \vee (\varphi(i) \wedge (\varphi \mathrm{U} \psi)(i+1))$. Hence, if $\varphi(i) = \bot$ then $(\varphi \mathrm{U} \psi)(i) = \psi(i)$ and if $\varphi(i) = \top$ then $(\varphi \mathrm{U} \psi)(i) = \psi(i) \vee (\varphi \mathrm{U} \psi)(i+1)$. Now, $(\chi_{3,4} \mathrm{U} r)(1) = a, (\chi_{3,4} \mathrm{U} r)(2) = b$ and $(\chi_{3,4} \mathrm{U} r)(5, 6, 7) = c$. Also, $(\chi_{3,4} \mathrm{U} r)(4) = r(4) \vee (\chi_{3,4} \mathrm{U} r)(5) = b \vee c$. Finally, $(\chi_{3,4} \mathrm{U} r)(3) = r(3) \vee (\chi_{3,4} \mathrm{U} r)(4) = b \vee (b \vee c) = b \vee c$. So $\chi_{3,4} \mathrm{U} r = (a, b, b \vee c, b \vee c, c, c, c)$. Performing a similar calculation backwards, we get $\chi_{4,5} \mathrm{S} (\chi_{3,4} \mathrm{U} r) = (a, b, b \vee c, b \vee c, b \vee c, c, c)$ which gives the value of block e in Figure 4.4 and leaves other blocks unchanged.

In general, denote the type of $\alpha_{i,j}$ by τ and the left and the right endpoint of $v(i, j)$ by l and r, respectively. Then $\varphi_{i,j}$ is constructed as follows:

- If $\tau = \text{ONE}$ then $\varphi_{i,j}(X) = \chi_{l,r} \vee X$.
- If $\tau = \text{ZERO}$ then $\varphi_{i,j}(X) = (\neg \chi_{l,r}) \wedge X$.
- If $\tau = \text{ID}$ then $\varphi_{i,j}(X) = X$.
- If $\tau = \text{OR}$ then $\varphi_{i,j}(X) = \chi_{l+1,r} \mathrm{S} (\chi_{l,r-1} \mathrm{U} X)$.
- If $\tau = \text{AND}$ then $\varphi_{i,j}(X) = (\neg \chi_{l+1,r}) \mathrm{T} ((\neg \chi_{l,r-1}) \mathrm{R} X)$.

It can be shown that the formula context $\varphi_{i,j}$ updates the block $v(i, j)$ and leaves the other blocks unchanged. Formally, for each layer C_i let $r_i \in \mathbb{B}^n$ be a proposition such that $r_i(k) = \top$ if $k \in v(i, j)$ for some j and $\alpha_{i,j}$ evaluates to $\top$ and $r_i(k) = \bot$, otherwise.

Lemma 4.4.2. *Fix $i > 0, j$ and let $p \in \mathbb{B}^n$ be any proposition that agrees with r_{i-1} on $v(i, j)$. Then $\varphi_{i,j}(p)$ is constant on $v(i, j)$ and equals the output of $\alpha_{i,j}$ and $\varphi_{i,j}(p)$ is equal to p elsewhere.*



Proof.

- **Case $\tau = \mathbf{ONE}$** Since $\bot \vee x = x$ and $\top \vee x = \top$, it holds that

$$(\chi_{l,r} \vee p)(i) = \begin{cases} \top \vee p(i) = \top & \text{if } i \leq l \leq r \\ \bot \vee p(i) = p(i) & \text{otherwise} \end{cases}$$

- **Case $\tau = \mathbf{ZERO}$** Similar to above, using $\top \wedge x = x$ and $\bot \wedge x = \bot$.
- **Case $\tau = \mathbf{ID}$** Trivial as $\varphi_{i,j}$ is the identity formula context.
- **Case $\tau = \mathbf{OR}$** Using the equality $r \,\mathrm{U}\, s = s \vee (r \wedge \mathrm{X}(r \,\mathrm{U}\, s))$, it follows that $(\chi_{l,r-1} \,\mathrm{U}\, p)(i) = p(i)$ for $i \notin [l, r-1]$ as $\chi_{l,r-1}(i) = \bot$. In particular, $(\chi_{l,r-1} \,\mathrm{U}\, p)(r) = p(r)$.

  For $i \in [l, r-1]$, it holds that $(\chi_{l,r-1}\mathrm{U}p)(i) = p(i) \vee (\bot \wedge (\chi_{l,r-1}\mathrm{U}p)(i+1)) = p(i) \vee (\chi_{l,r-1}\mathrm{U}p)(i+1)$. By induction, we have $(\chi_{l,r-1}\mathrm{U}p)(j) = p(j) \vee p(j+1) \vee \cdots \vee p(r)$ for $j \in [l, r]$.

  Similarly, for any proposition s, it holds that

$$(\chi_{l+1,r} \,\mathrm{S}\, s)(j) = \begin{cases} s(l) \vee s(l+1) \vee \cdots \vee s(j) & \text{if } j \in [l, r] \\ s(j) & \text{otherwise} \end{cases}$$

  Putting the above equalities together, we obtain that $\varphi_{i,j}(p)(k) = (\chi_{l+1,r} \,\mathrm{S}\, (\chi_{l,r-1} \,\mathrm{U}\, p))(k)$ equals

$$(\chi_{l+1,r} \,\mathrm{S}\, (\chi_{l,r-1} \,\mathrm{U}\, p))(k) = \begin{cases} p(l) \vee p(l+1) \vee \cdots \vee p(r) & \text{if } k \in [l, r] \\ p(k) & \text{otherwise} \end{cases}$$

- **Case $\tau = \mathbf{AND}$** This case is dual to the one above. Using the equality $r \,\mathrm{R}\, s = s \wedge (r \vee \mathrm{X}(r \,\mathrm{R}\, s))$, it follows that $((\neg\chi_{l,r-1}) \,\mathrm{R}\, p)(i) = p(i)$ for $i \notin [l, r-1]$ as $(\neg\chi_{l,r-1})(i) = \top$. In particular, $((\neg\chi_{l,r-1}) \,\mathrm{R}\, p)(r) = p(r)$.

  For $i \in [l, r-1]$, it holds that $((\neg\chi_{l,r-1}) \,\mathrm{R}\, p)(i) = p(i) \wedge (\top \vee ((\neg\chi_{l,r-1}) \,\mathrm{R}\, p)(i+1) = p(i) \wedge ((\neg\chi_{l,r-1}) \,\mathrm{R}\, p)(i+1)$. By induction, we have $((\neg\chi_{l,r-1}) \,\mathrm{R}\, p)(j) = p(j) \wedge p(j+1) \wedge \cdots \wedge p(r)$ for $j \in [l, r]$.

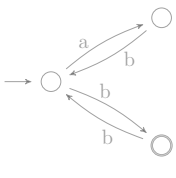
  Similarly, for any proposition s, it holds that

$$((\neg\chi_{l+1,r}) \,\mathrm{T}\, s)(j) = \begin{cases} s(l) \wedge s(l+1) \wedge \cdots \wedge s(j) & \text{if } j \in [l, r] \\ s(j) & \text{otherwise} \end{cases}$$

  Putting the above equalities together, we obtain that $\varphi_{i,j}(p)(k) = ((\neg\chi_{l+1,r}) \,\mathrm{T}\, ((\neg\chi_{l,r-1}) \,\mathrm{R}\, p))(k)$ equals

$$((\neg\chi_{l+1,r}) \,\mathrm{T}\, ((\neg\chi_{l,r-1}) \,\mathrm{R}\, p))(k) = \begin{cases} p(l) \wedge p(l+1) \wedge \cdots \wedge p(r) & \text{if } k \in [l, r] \\ p(k) & \text{otherwise} \end{cases}$$

□



By combining all formula contexts for the gates in the i-th layer, we have that the formula context $\psi_i(X) = \varphi_{i,1} \circ \varphi_{i,2} \circ \cdots \circ \varphi_{i,n_i}$ evaluates the i-th layer C_i of C. Therefore, the formula $\varphi = (\psi_k \circ \psi_{k-1} \circ \cdots \circ \psi_1)(r_0)$ computes the output of the circuit.

Lemma 4.4.3. *Let ψ_i, φ be as above. Then $\psi_i(r_{i-1}) = r_i$ and $\varphi(r_0)(1) = \top$ if and only if C evaluates to $\top$. Moreover, φ can be built in* L.

Proof. Let $\gamma_j = \varphi_{i,j} \circ \varphi_{i,2} \circ \cdots \circ \varphi_{i,n_i}$. We shall prove by downward induction that $\gamma_j(r_{i-1})$ equals r_i on $v(i,j) \cup \cdots \cup v(i,n_i)$ and r_{i-1} elsewhere.

- **Case $j = n_i$** Immediate from Lemma 4.4.2 applied to φ_{i,n_i}.
- **Case $j < n_i$** Let $\pi = \gamma_{j+1}(r_{i-1})$. Now, $v(i,t)$'s partition the trace, Lemma 4.4.4, and, Lemma 4.4.2, the value of $\varphi_{i,j}$ on $v(i,j)$ depends only on the trace segment $v(i,j)$ and all other trace values are left unaffected. By the induction hypothesis, π agrees with r_{i-1} on $v(i,j)$. Thus, $\varphi_{i,j}(\pi)$ equals r_i on $v(i,j)$ and r_{i-1} everywhere else. By the induction, $\varphi_{i,j}(\pi)$ equals r_i on $v(i,j) \cup \cdots \cup v(i,n_i)$ and r_{i-1} elsewhere as required.

Finally, by induction, it is easily seen that $\varphi = (\psi_k \circ \psi_{k-1} \circ \cdots \circ \psi_1)(r_0) = r_n$. Thus $r_0 \models \varphi$ if and only if $r_n(1) = \top$ which is precisely when the output gate of the circuit evaluates to true. □

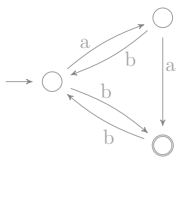
This finishes the construction of an LTL formula mimicking evaluation of upward layered monotone circuits C provided we are given a valid partition.

4.4.0.1 Constructing a Valid Partitioning

In this section, we show how to devise $v(i,j)$'s – the partitioning of the trace. Recall (Property 4.4.1) the two conditions relating a partitioning to the underlying circuit that every valid partitioning has to satisfy.

The partitioning is constructed as follows. First, without loss of generality, connecting to a gate in the previous layer if necessary, we assume that all ONE and ZERO gates not in C_0 have at least one predecessor.

Then, recall that $\alpha_{i,j}$ is the j-th gate in the i-th layer of C. So given a gate $\alpha_{i,j}$ there is unique rightmost gate in the layer C_{i+1} that $\alpha_{i,j}$ is connected to by a wire. Now, start at $\alpha_{i,j}$ and take the rightmost wires until the sink is reached. Denote the traversed path by π_u. Similarly, there is unique rightmost gate in the layer C_{i-1} that $\alpha_{i,j}$ is connected to by a wire. Start at $\alpha_{i,j}$ and take the rightmost wires going



down until a gate in C_0 is reached. Denote the traversed path by π_d. Let π be the concatenation of π_d and π_u. (See Figure 4.4)

Let $k_{i,j}$ be the number of wires to the left of π. A wire from $\alpha_{i,j}$ to $\alpha_{i+1,k}$ is to the left of the wire from $\alpha_{i,a}$ to $\alpha_{i+1,b}$ if $j < a$ or $k < b$. We store the output of gate $\alpha_{i,j}$ in the block $v(i,j) := [k_{i,j-1} + 1, k_{i,j} + 1]$. We use $k_{i,0} = 0$.

Figure 4.4 shows a circuit and the partitioning obtained by the above procedure. For example, the rightmost wire going up and down from e are $e \to g$ and $c \to e$, respectively. Thus, $\pi_u = e \to g$ and $\pi_d = c \to e$. The path $\pi = c \to e \to g$ is highlighted in the figure. Four wires $a \to d, b \to d, b \to e, d \to g$ are to the left of π. We associate the block $[3, 5]$ with gate e. All blocks, grouped by layers, are shown in Figure 4.4.

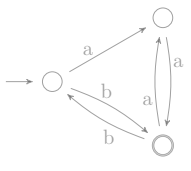
The following lemma summarises the important properties of $k_{i,j}$'s.

Lemma 4.4.4. *Let $k_{i,j}$'s and $v(i,j)$'s be as above. Then the following hold:*

- *$k_{i,j-1} < k_{i,j}$ for every i and j,*
- *$k_{i,n_i} = k_{j,n_j}$ for every i and j,*
- *$k_{i,n_i} \leq m$ for every i,*
- *for every i and $j = 1, \ldots, n_i$ the intervals $v(i,j)$'s partition $[1, k_{i,n_i}]$,*
- *if $\alpha_{i+1,j}$ depends on $\alpha_{i,p}, \alpha_{i,p+1}, \ldots, \alpha_{i,q}$ then $v(i+1,j) \subseteq \cup_{r=p,\ldots,q} v(i,r)$ and $v(i+1,j)$ overlaps with each $v(i,r)$ for $p \leq r \leq q$,*
- *each $k_{i,j}$ can be computed in L.*

Proof. Note that, due to planarity of the underlying circuit, each $\alpha_{i,j}$ depends on the consecutive block of gates one layer below and that $\alpha_{i,j}$ and $\alpha_{i,j+1}$ share at most one predecessor.

- $\boldsymbol{k_{i,j-1} < k_{i,j}}$ Note that for any i, j, the rightmost predecessor (successor) of $\alpha_{i,j}$ is always to the right in the planar embedding to the rightmost predecessor (successor) of $\alpha_{i,j-1}$. Let π be the path constructed by taking the rightmost wires starting from $\alpha_{i,j}$. In particular, the wire from $\alpha_{i,j-1}$ to its rightmost predecessor is strictly to the left of π. Thus, $k_{i,j-1} < k_{i,j}$ as necessary.
- $\boldsymbol{k_{i,n_i} = k_{j,n_j} \leq m}$ **for every $\boldsymbol{i}$ and $\boldsymbol{j}$.** Since the rightmost gate can only be a predecessor of the rightmost gate one layer higher, it follows that the path π is equal for all rightmost gates in all layers.



- **for every i and $j = 1, \ldots, n_i$ the intervals $v(i,j)$'s partition $[1, k_{i,n_i}]$.** By the first part, all $v(i,j)$'s in the same layer are disjoint. By construction, they cover $[1, k_{i,n_i}]$ entirely.
- **if $\alpha_{i+1,j}$ depends on $\alpha_{i,p}, \alpha_{i,p+1}, \ldots, \alpha_{i,q}$ then**

$$v(i+1,j) \subseteq \bigcup_{r=p,\ldots,q} v(i,r)$$

and $v(i+1,j)$ overlaps with each $v(i,r)$ for $p \leq r \leq q$. Construct the paths by taking the rightmost wires from $\alpha_{i+1,j-1}, \alpha_{i+1,j}, \alpha_{i,p}$ and $\alpha_{i,q}$ and denote them by π_{j-1}, π, π_p and π_q respectively. Notice that π and π_p coincide on the layers above $i+1$ and that π and π_q coincide on the layers below i. On the other hand, π_q is always to the right of π_p. Thus, $k_{i,p} < k_{i+1,j} \leq k_{i,q}$. Also, π_{j-1} is always to the left of π_p. Thus $k_{i+1,j-1} < k_{i,p}$ as required.

- **each $k_{i,j}$ can be computed in L.** The algorithm needs to keep track of one counter: number of wires strictly to the left, and the current gate. By traversing the list of wires, it is easy to calculate the rightmost wire going up (down). By another pass through the list of wires, the counter is incremented by the wires to the left in the current layer. □

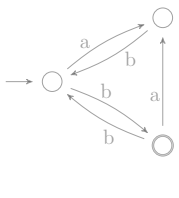
This finishes the construction of a valid partitioning for upward layered circuits. In the previous section who showed how to construct an LTL formula given a valid partitioning. Putting the two together we obtain a reduction from upward-layered CVP to LTL path checking. It was shown in [KF09] that the latter is in $\mathsf{AC}^1[\mathsf{logDCFL}]$. Therefore:

Theorem 4.4.5. *The CVP for upward-layered monotone circuits is in* $\mathsf{AC}^1[\mathsf{logDCFL}]$.

An alternative proof of Theorem 4.4.5 already appeared in [LMS06]. Moreover, the relationship shows that any improvement in LTL path checking would entail an improvement in the evaluation of upward-layered monotone circuits.

The above reduction assumes the monotonicity of the input circuit. However, if the target logic LTL is extended to include binary exclusive or (xor) as a connective, then evaluating NOT gates becomes possible using $\varphi_{i,j}(X) = \chi_{l,r} \oplus X$ as a formula context for NOT gate $\alpha_{i,j}$.

Lemma 4.4.6. *Let $\varphi_{i,j}(X) = \chi_{l,r} \oplus X$. Then $\varphi_{i,j}(p)(i) = p(i)$ if $i \notin [l,r]$ and $\varphi_{i,j}(p)(i) = \neg p(i)$ if $i \in [l,r]$*



Proof. Since $x \oplus \top = \neg x$ and $x \oplus \bot = x$ the result follows. □

Noting that CVP is P-complete for general (non-monotone) upward stratified circuits [Gol77], we thus have the following:

Theorem 4.4.7. $\mathsf{LTL} + \mathsf{Xor}$ *path checking is* P*-complete.*

Thus, the complexity of LTL path checking depends on the monotonicity of the Boolean connectives present in the formula.

4.5 MTL

In the previous section we showed a hardness result for LTL path checking. In this and the following sections we study and give path-checking algorithms for various temporal logics.

We begin with MTL. Specifically, we show how the tree-contraction method of [KF09] extends to full MTL; giving an $\mathsf{AC}^1[\mathsf{logDCFL}]$ path-checking algorithm for MTL. By [KF09], summarised in Section 4.1, it suffices to construct upward stratified transducer circuits for U_I and its duals. That is, given a Boolean vector s it suffices to construct transducer circuits C and C' such that for any $x \in \mathbb{B}^n$ we have $C(x) = s\ \mathrm{U}_I\ x$ and $C'(x) = x\ \mathrm{U}_i\ s$ and similarly for the dual temporal operators.

Let π be the input trace with (floating-point) timestamps $t_1, \ldots, t_n$. Fix an interval I and consider the U_I operator. We now describe a dynamic-programming approach that yields planar circuits calculating $(\psi_1\ \mathrm{U}_I\ \psi_2)(i)$.

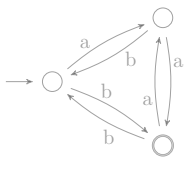
For $i \neq j$ the values $(\psi_1\ \mathrm{U}_I\ \psi_2)(i)$ and $(\psi_1\ \mathrm{U}_I\ \psi_2)(j)$ depend on the values of subformulae in some future intervals. In general, these intervals overlap and so naive constructions of transducer circuits are not planar. See Figures 4.5 and 4.6 for the kind of circuits we build.

Let $s \in \mathbb{B}^n$ be a vector. We construct circuits for $s\ \mathrm{U}_I\ \varphi$ and $\varphi\ \mathrm{U}_I\ s$ for known s. First consider the case $s\ \mathrm{U}_I\ \varphi$. (see Figure 4.5)

For index $1 \leq i \leq n$ the formula $(s\ \mathrm{U}_I\ \varphi)(i)$ is true if there is $j \geq i$ such that $t_j \in t_i + I$ and $\varphi(j) = \top$ and $s(k) = \top$ for all $i \leq k < j$. So let $T_i = \{j \mid t_j \in t_i + I\}$ be the set of indices of timestamps in $t_i + I$. If $T_i = \emptyset$ then $(s\ \mathrm{U}_I\ \varphi)(i) = \bot$.

Otherwise, let $\mathrm{first}(i) = \min T_i$ and $\mathrm{last}(i) = \max T_i$ be the first and the last index in the interval $t_i + I$, respectively. So $(s\ \mathrm{U}_I\ \varphi)(i)$ is true if there exists $\mathrm{first}(i) \leq j \leq \mathrm{last}(i)$ such that $\varphi(j) = \top$ and $s(k) = \top$ for all $i \leq k < j$.

Now, the value of s is known. So let $\mathrm{seg}(i) = \min\{j \mid j \geq i \wedge s(j) = \bot\}$ be the first index no smaller than i such that $s(j)$ evaluates to false, i.e., $s(j)$ is true from i



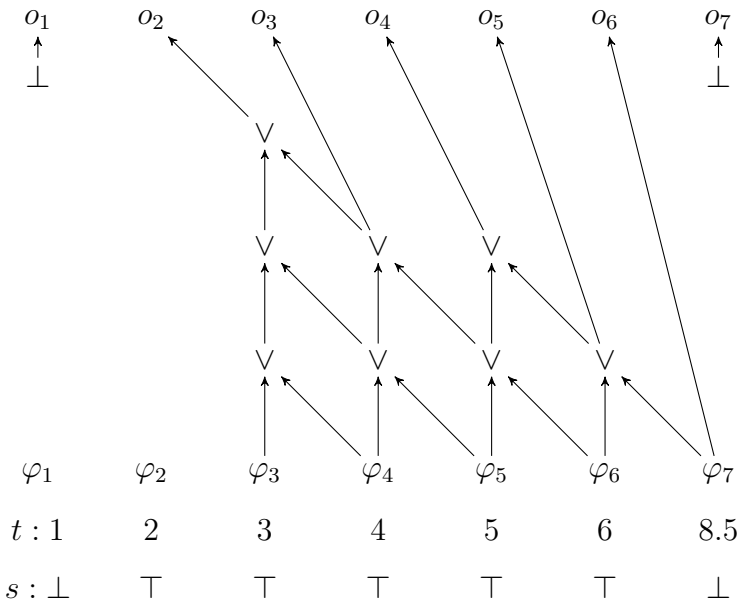


Figure 4.5: Transducer circuit for $s \,\mathrm{U}_{[1,5]}\, \varphi$ The first line below the circuits are timestamps, the second row are values of s. The inputs and the outputs of the circuit are denoted φ_i and o_i respectively.

to $\mathrm{seg}(i) - 1$. Thus, $(s \,\mathrm{U}_I\, \varphi)(i)$ is true if there exists $\mathrm{first}(i) \leq j \leq \mathrm{last}(i)$ such that $\varphi(j) = \top$ and $j \leq \mathrm{seg}(i)$. So take $L_i = \mathrm{first}(i)$ and $R_i = \min(\mathrm{last}(i), \mathrm{seg}(i))$. Then

$$(s \,\mathrm{U}_I\, \varphi)(i) \iff \bigvee_{L_i \leq j \leq R_i} \varphi(j)$$

To build the circuits, we formalise the intuition from Figure 4.5. The circuit C consists of internal gates $d_{p,q}$ and output gates o_i for each $1 \leq i \leq n$. Each internal gate $d_{p,q}$ calculates $\varphi_p \vee \cdots \vee \varphi_q$. Precisely, $d_{p,q}$ is present in the circuit if there is an i such that $L_i \leq p \leq q \leq R_i$. If $p = q$ then $l(d_{p,q}) = (\mathrm{ID}, \varphi_p)$. Otherwise, $l(d_{p,q}) = (\mathrm{OR}, d_{p,q-1}, d_{p+1,q})$.

For the output gates, we define o_i so that

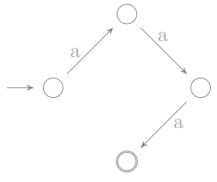
$$o_i = \bigvee_{L_i \leq j \leq R_i} \varphi(j) = (s \,\mathrm{U}_I\, \varphi)(i).$$

Specifically, if $T_i = \emptyset$ then we set $l(o_i) = \bot$, otherwise, $l(o_i) = (\mathrm{ID}, d_{L_i,R_i})$.

An embedding $\gamma : C \to \mathbb{R}^2$ for the circuit C is

$$\gamma(o_i) = (i, 2n^2) \quad \gamma(\varphi_i) = (i, 0) \quad \gamma(d_{p,q}) = (p, q - p + 1).$$

Observe that $L_i \leq L_{i+1}$ and $R_i \leq R_{i+1}$. Hence, it cannot happen that $L_i < L_j \leq R_j < R_i$ for some i and j. So the intervals may overlap but never is one properly contained in another. This ensures that the embedding is planar.



Lemma 4.5.1. *The embedding γ is planar.*

Proof. There are three types of edges in the embedding induced by γ.

- edges between φ_i and $d_{p,q}$ gates,
- edges between two $d_{p,q}$ gates,
- edges between an $d_{p,q}$ gate and an output gate.

Clearly, the edges of the first type do not intersect. Also, by considering the y coordinate, the edges of the first type do not intersect with the edges of the second or third type.

We call the edges of the second type the *internal* edges and the edges of the third type the *external* edges. We now prove by case analysis that internal and/or external edges do not intersect. We use the following convention. If c is a gate then $\gamma(c).x$ and $\gamma(c).y$ denote the x and the y coordinate of $\gamma(c)$, respectively.

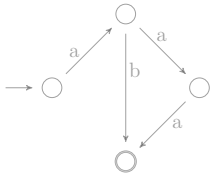
Internal/Internal. Note that γ maps each $d_{p,q}$ to a grid point. Further, every internal edge connects the point (x, y) with either the point above $(x, y + 1)$ or the diagonally opposite one $(x - 1, y + 1)$. Since each internal edge is inside unit square and only one type of diagonals is present it follows that two internal edges never intersect.

External/External. Denote the edge $\gamma(d_{L_i,R_i}) \to \gamma(o_i)$ by E_i. For $i < j$ note that $\gamma(o_i).x < \gamma(o_j).x$. Recall that $L_i = \text{first}(i) = \min T_i = \min\{j \mid t_j \in t_i + I\}$. Hence, $L_i \leq L_j$ and so $\gamma(d_{L_i,R_i}).x = L_i \leq L_j = \gamma(d_{L_j,R_j}).x$. It follows that the edges E_i and E_j do not intersect.

Internal/External. Note that for any $d_{p,q}$ gate in the circuit we have $0 \leq p \leq q \leq n$ and so $\gamma(d_{p,q}).y \leq n$. On the other hand, $\gamma(o_i).y = 2n^2$. So suppose that there is an intersection (x, y) between some internal edge I and some external edge E_i. Then $\gamma(d_{L_i,R_i}).y < y \leq n$. Since the slope of E_i is always at least $\frac{2n^2-n}{n} = 2n - 1$, it follows that $x \in (L_i - 1, L_i + 1)$. But that means that (at least) one end point of I has the x-coordinate equal to L_i. That is, there is a gate $d_{p',q'}$ such that $\gamma(d_{p',q'}) = (L_i, y)$. By the construction, there must be an index j such that $L_j \leq p' \leq q' \leq R_j$.

Hence, the point $\gamma(d_{p',q'})$ is above $\gamma(d_{L_i,R_i})$ and so, by the construction of γ, it follows that $q' > R_i$. Hence, we have $L_j \leq p' = L_i \leq R_i < q' \leq R_j$. But by the remark before the statement of this lemma, this is impossible. □

Finally, note that each L_i and R_i are numbers in $\{1, \ldots, n\}$ and hence only $\log n$ bits are needed to store them. It is easy to see that it is possible to compute L_i and



R_i for every i in logarithmic space[6]. Hence, the circuit construction can be carried out in logarithmic space.

Lemma 4.5.2. *Let p be any proposition. For each i, set the input φ_i of the circuit to $p(i)$. Then for each j, the value of o_j is true if and only if $(s \,\mathrm{U}_I\, p)(j)$ is true.*

Proof. One can easily show by induction on $q - p$ that $d_{p,q} = \varphi_p \vee \cdots \vee \varphi_q$. As noted in the main text,

$$\pi, i \models s \,\mathrm{U}_I\, \varphi \text{ if and only if } \bigvee_{L_i \le j \le R_i} \pi, j \models \varphi \text{ is true.}$$

The result now immediately follows. □

We now give an analogous derivation and circuit construction for $\varphi \,\mathrm{U}_I\, s$. See Figure 4.6 for an example of a resulting circuit.

For index $1 \le i \le n$ the formula $(\varphi \,\mathrm{U}_I\, s)(i)$ is true if there exists $j \ge i$ such that $t_j \in t_i + I$ and $s(j) = \top$ and $\varphi(k) = \top$ for all $i \le k < j$. Since s is known, we choose the first possible j. So let $\mathrm{limit}(i) = \min\{j \mid \mathrm{first}(i) \le j \le \mathrm{last}(i) \wedge s(j) = \top\}$ be the first j in the interval $t_i + I$ such that $s(j)$ is true.

If there is no such index then $(\varphi \,\mathrm{U}_I\, s)(i) = \bot$. Otherwise, $(\varphi \,\mathrm{U}_I\, s)(i)$ is true if $\varphi(k) = \top$ for all $i \le k < \mathrm{limit}(i)$. That is,

$$(\varphi \,\mathrm{U}_I\, s)(i) \iff \bigwedge_{i \le j < \mathrm{limit}(i)} \varphi(j).$$

Now, the circuit C (see Figure 4.6) consists of gates $c_{p,q}$ calculating $\varphi_p \wedge \cdots \wedge \varphi_q$ and output gates o_i for $i = 1 \ldots n$. The gate $c_{p,q}$ is present in C if there is i such that $i \le p \le q < \mathrm{limit}(i)$. If $p = q$ then $l(c_{p,q}) = (\mathrm{ID}, \varphi_p)$. Otherwise, $l(c_{p,q}) = (\mathrm{AND}, c_{p,q-1}, c_{p+1,q})$.

For output, we set o_i so that

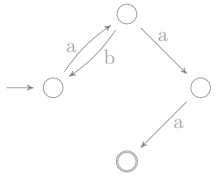
$$o_i = \bigwedge_{i \le j < \mathrm{limit}(i)} \varphi_j = (\varphi \,\mathrm{U}_I\, s)(i).$$

If $\mathrm{limit}(i) = \infty$ then $l(o_i) = \bot$, if $\mathrm{limit}(i) = i$ then $l(o_i) = \top$ and else $l(o_i) = (\mathrm{ID}, c_{i,\mathrm{limit}(i)-1})$.

The embedding $\gamma : C \to \mathbb{R}^2$ of the circuit C is the same as above,

$$\gamma(o_i) = (i, 2n^2) \quad \gamma(\varphi_i) = (i, 0) \quad \gamma(c_{p,q}) = (p, q - p + 1).$$

[6] $t_j - t_i \ge a$ can be checked by first comparing the largest bits. If they are equal then by comparing the second largest bits and so on.



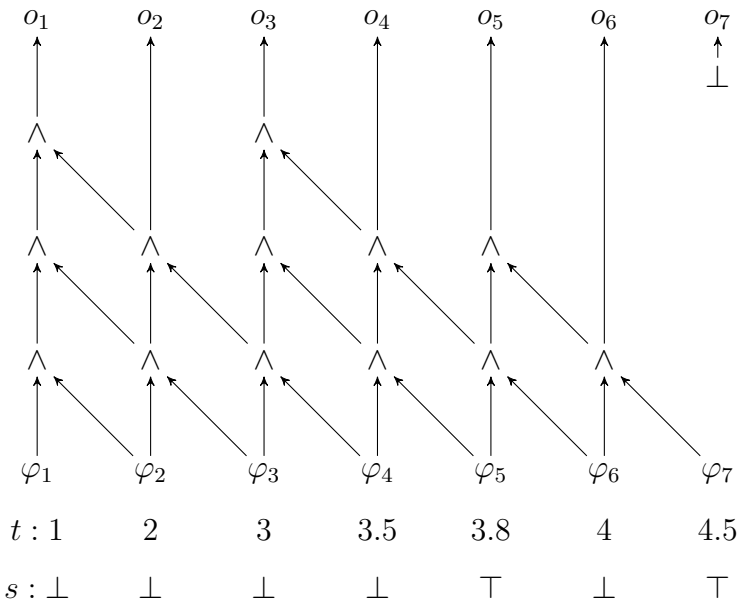


Figure 4.6: Transducer circuit for $\varphi \, \mathrm{U}_{[1,5]} \, s$. The first line below the circuits are timestamps, the second row are values of s. The inputs and the outputs of the circuit are denoted φ_i and o_i respectively.

Since, $i < j$ implies $\mathrm{limit}(i) \leq \mathrm{limit}(j)$, as in the previous case (Lemma 4.5.1), the embedding is planar.

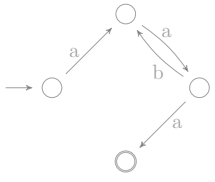
This finishes the construction of circuits for U_I. Circuits for the dual operators of U_I are obtained either by dualising OR and AND gates (Release operator), by performing the construction backwards in time (Since operator) or both (Trigger operator). Therefore,

Theorem 4.5.3. MTL *path checking is in* $\mathsf{AC}^1[\mathsf{logDCFL}]$.

A weaker result appeared in [KF12], where the authors gave an $\mathsf{AC}^1[\mathsf{logDCFL}]$ algorithm for an extension of LTL where temporal operators can be labelled by intervals of the form $[0, a]$. In other words, an $\mathsf{AC}^1[\mathsf{logDCFL}]$ algorithm in [KF12] works for a fragment of MTL interpreted over traces with integral timestamps $t_i = i$ and intervals of the form $[0, a]$ for $a \in \mathbb{N}$.

4.6 UTL

An examination of the algorithmic constructions of [KF09] as well as the construction in previous section shows that the most intricate circuits arise in handling the Until operator. In particular, the circuits for LTL operators $s \, \mathrm{U} \, \psi$ and $\psi \, \mathrm{U} \, s$ are not uniform but depend on s.



In this section we devise an AC^1 tree-contraction algorithm for Unary Temporal Logic (UTL)—a temporal logic obtained by removing binary temporal operators from LTL. Restricting to UTL enables us to avoid the circuits and represent all intermediate functions in a more concise way.

The algorithm presented in this section works even if XOR Boolean connective is allowed and it is based on the analysis of functions arising in the tree contraction algorithm applied to UTL formulae.

In the rest of the section, we make use of the following notation. A vector $v \in \mathbb{B}^n$ is *downward monotone* if $v(i+1) = \top \implies v(i) = \top$. It is *upward monotone* if $v(i-1) = \top \implies v(i) = \top$. A vector is *monotone* if it is upward or downward monotone. The set of monotone vectors is denoted by $\mathbb{M}$.

Given a UTL formula φ and a trace π, we now describe a tree contraction algorithm for determining whether $\pi, 1 \models \varphi$. The tree used in the tree-contraction algorithm is the parse tree of the given UTL formula φ.

First consider the future-only fragment of UTL. A UTL formula consists of Boolean connectives and unary temporal operators F and G. We shall show how to represent and manipulate in AC^0 functions corresponding to the partial evaluation of these.

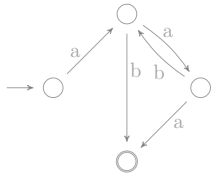
We begin with temporal operators. Let $p \in \mathbb{B}^n$ be any proposition. If $p(i) = \bot$ for every i then $(\mathrm{F}p)(i) = \bot$ for every i. Otherwise, let i be the largest index such that $p(i) = \top$. Then, $(\mathrm{F}p)(j) = \top$ for all $j \leq i$. By construction, $p(k) = \bot$ for all $k > i$. Hence, $(\mathrm{F}p)(k) = \bot$ for all $k > i$. Thus, $\mathrm{F}p$ is downward monotone and depends only on the largest i with $p(i) = \top$. In particular, only $n+1$ possible values exist for $\mathrm{F}p$.

Similarly, let t be the largest index such that $p(t) = \bot$. Then $p(j) = \top$ for all $j > t$. Hence $(\mathrm{G}p)(j) = \top$ for all $j > t$. Since $p(t) = \bot$ we have $(\mathrm{G}p)(k) = \bot$ for all $k \leq t$. Thus, $\mathrm{G}p$ is upward monotone and depends only on the largest t with $p(t) = \bot$. In particular, only $n+1$ possible values exist for $\mathrm{G}p$. Thus, we have shown[7]:

Lemma 4.6.1. *For any Boolean vector $x \in \mathbb{B}^n$ the vectors $\mathrm{F}x$ and $\mathrm{G}x$ are monotone.*

Therefore, for any formula ψ the value of $\mathrm{F} \circ \psi$ or $\mathrm{G} \circ \psi$ is a monotone vector, of which there are only $2n$ many. Hence for any formula context $\varphi(X)$, the formula contexts $\varphi \circ (\mathrm{F}X)$ and $\varphi \circ (\mathrm{G}X)$ can be represented as $g \circ F$ or $g \circ G$ where $g : \mathbb{M} \to \mathbb{B}^n$ is a *function with monotone domain*. Since $|\mathbb{M}| = O(n)$, enumerating all outputs of g explicitly requires only $|g| = O(n^2)$ space instead of $O(n2^n)$ required to enumerate all possibilities.

[7]Similar results hold for the past equivalents of G and F.



For the Boolean operators, notice that they are applied componentwise and obey the usual identities:

$$\begin{array}{lll} \bot \wedge p = \bot & \bot \vee p = p & \bot \oplus p = p \\ \top \wedge p = p & \top \vee p = \top & \top \oplus p = \neg p \end{array}$$

Therefore, to represent partial evaluation of conjunction $(p \wedge X, x \wedge X)$, disjunction $(p \vee X, X \vee p)$ and xor $(p \oplus X, X \oplus p)$ it suffices to keep track whether each component is $\bot, \top$ or equal to the original or the negation of the value in X.

Furthermore, Next (Xp) and Yesterday (Yp) temporal operators shift p by 1 and -1, respectively. Let m be the size of the input formula. The last two paragraphs motivate the definition of filters: let $v \in \{\bot, \top, \mathrm{ID}, \mathrm{NOT}\}^n$ and $k \in [-m, m]$ satisfy $v(i) \in \mathbb{B}$ if $i + k \notin \{1, \ldots, n\}$. Then a *filter with offset k and pattern v* is the function $f_{v,k} : \mathbb{B}^n \to \mathbb{B}^n$ such that

$$f_{v,k}(p)(i) = \begin{cases} \bot & \text{if } v(i) = \bot \\ \top & \text{if } v(i) = \top \\ p(i+k) & \text{if } v(i) = \mathrm{ID} \\ \neg p(i+k) & \text{if } v(i) = \mathrm{NOT} \end{cases}$$

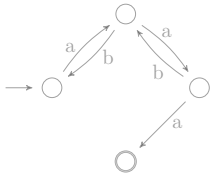
For each component a filter keeps track of whether it is $\bot, \top$, negated or the original value. A filter can express basic expressions. The identity function as well as the partial evaluation of conjunction, disjunction, and xor are expressible as filters with offset 0 and v from $\{\bot, \mathrm{ID}\}^n, \{\top, \mathrm{ID}\}^n, \{\mathrm{ID}, \mathrm{NOT}\}^n$, respectively. Temporal operators Next and Yesterday are identity filters with offsets 1 and -1, respectively.

To combine the representations of Boolean and temporal operators, we represent the functions arising in the tree-contraction algorithm as follows. If the contracted subtree S does not contain F or G operators then it is representable by a filter. If it contains F or G then let T be the first such occurrence. Then the segment from the leaves to T is representable by a filter and the segment above T is representable by a function with monotone domain. Thus, the function h associated with S can be represented as:

$$h = \begin{cases} \text{filter} & \text{no temporal operator} \\ f \circ T \circ \text{filter} & T \text{ is the first temporal operator; } f : \mathbb{M} \to \mathbb{B}^n \end{cases}$$

That is, the function is either a filter or a filter followed by a unary temporal operator and a function with monotone domain. The choice corresponds to the presence of a temporal operator in the contracted subtree. We call functions of this form the *assembled* functions.

Such functions can be stored efficiently. For a filter $f_{v,k}$, storing v explicitly and k in unary requires $O(n + |\varphi|)$ bits per filter. Precisely, we store filter $f_{v,k}$ as follows.



- For each admissible value of k we store a Boolean variable f_k which is true if and only if the offset of f is k.
- We keep n pairs of Boolean variables, each pair encoding the value of some $v(i)$.

Further, a function with monotone domain $g : \mathbb{M} \to \mathbb{B}^n$ can be stored as a look-up table on $O(n)$ entries. For each monotone vector $m \in \mathbb{M}$ and index $1 \leq i \leq n$ we store a variable $g_{m,i}$ which is true if and only if $g(m)(i)$ is true.

Moreover, the assembled functions can be used in a tree-contraction algorithm for UTL (see Section 4.1 for definition). Each contraction step consists of three substeps and we show that each can be implemented in AC^0.

- Evaluation $c = f_1(l)$ is in AC^0 by Lemma 4.6.2.

  Lemma 4.6.2. *Let h be an assembled function and $x \in \mathbb{B}^n$. Then for every index $1 \leq i \leq n$ the value $h(x)(i)$ can be calculated in AC^0.*

  Proof. We prove this in three stages, one stage for each possible part of h.

  If $f_{v,k}$ is a filter, recall that we use variables f_k to denote whether the offset is k. Then $f_{v,k}(x)(i)$ is true if and only if

$$\bigvee_{k,1 \leq i+k \leq n} \left\{ f_k \wedge \left[\begin{array}{l} (v(i+k) = \top) \vee \\ (v(i+k) = \mathrm{ID} \ \wedge \ x(i+k)) \vee \\ (v(i+k) = \mathrm{NOT} \ \wedge \ \neg x(i+k)) \end{array} \right] \right\}$$

  If T is a temporal operator, there are two cases. If $T = \mathrm{F}$ we calculate

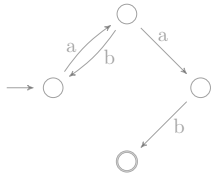
$$\mathrm{F}(x)(i) = \bigvee_{i \leq j \leq n} x(i).$$

  If $T = \mathrm{G}$ we calculate

$$\mathrm{G}(x)(i) = \bigwedge_{i \leq j \leq n} x(i).$$

  Finally, if f is a function with monotone domain, that recall that f is represented by a look-up table. Hence, assuming x is monotone, we calculate

$$f(x)(i) = \bigvee_{m \in \mathbb{M}} \Big[(x = m) \ \wedge \ f_{v,i} \Big]$$



where $x = m$ is just a shorthand for the equivalence of two n-bit vectors:

$$\bigwedge_{1 \leq j \leq n} (x(j) \iff m(j)).$$

Now, notice that each individual step is a constant-depth polynomial-size formula, thus a constant-depth polynomial-size circuit. Therefore, by composing the three parts together, we obtain an AC^0 circuit for assembled functions. □

- The functions $h = \lambda x.c \wedge x, \lambda x.c \vee x$ are easily representable as filters. If p is a temporal operator (F or G) then $h = \mathrm{ID} \circ \mathrm{F} \circ \mathrm{ID}$ or $h = \mathrm{ID} \circ \mathrm{G} \circ \mathrm{ID}$, which are both easily representable.
- Finally, we have to show how to compose assembled functions. For two assembled functions h, h', we have four cases for the composition $h \circ h'$ depending on the structure of h and h'.

  Note that filters are closed under composition and can be composed in AC^0.

  Lemma 4.6.3. *Let $f_{v,k}$ and $g_{w,l}$ be two filters then $f_{v,k} \circ g_{w,l}$ is a filter and the composition can be computed in AC^0.*

  Proof. Expanding the definition of $f_{v,k}$ we calculate

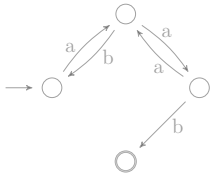
  $$(f_{v,k} \circ g_{w,l})(p)(i) = (f_{v,k}(g_{w,l}(p)))(i)$$

  as equal to

  $$(f_{v,k}(g_{w,l}(p)))(i) = \begin{cases} \bot & \text{if } v(i) = \bot \\ \top & \text{if } v(i) = \top \\ g_{w,l}(p)(i+k) & \text{if } v(i) = \mathrm{ID} \\ \neg q_{w,l}(p)(i+k) & \text{if } v(i) = \mathrm{NOT} \end{cases}$$

  Expanding the definition of $g_{w,l}$ we obtain:

  $$(f_{v,k}(g_{w,l}(p)))(i) = \begin{cases} \bot & \text{if } v(i) = \bot \\ \top & \text{if } v(i) = \top \\ \bot & \text{if } v(i) = \mathrm{ID} \text{ and } w(i+k) = \bot \\ \top & \text{if } v(i) = \mathrm{ID} \text{ and } w(i+k) = \top \\ p(i+k+l) & \text{if } v(i) = \mathrm{ID} \text{ and } w(i+k) = \mathrm{ID} \\ \neg p(i+k+l) & \text{if } v(i) = \mathrm{ID} \text{ and } w(i+k) = \mathrm{NOT} \\ \bot & \text{if } v(i) = \mathrm{NOT} \text{ and } w(i+k) = \bot \\ \top & \text{if } v(i) = \mathrm{NOT} \text{ and } w(i+k) = \top \\ p(i+k+l) & \text{if } v(i) = \mathrm{NOT} \text{ and } w(i+k) = \mathrm{ID} \\ \neg p(i+k+l) & \text{if } v(i) = \mathrm{NOT} \text{ and } w(i+k) = \mathrm{NOT} \end{cases}$$



which defines a filter with offset $k+l$. It is easy to see that the above expression gives rise to an AC^0 circuit for the composition of two filters. □

If h' is a filter then the composition is in AC^0 also by the above Lemma 4.6.3.

If $h' = f' \circ T' \circ \text{filter}'$ then if h is a filter then the result follows by Lemma 4.6.4, which is shown later. In the remaining case, write $h = f \circ T' \circ \text{filter}$, then we calculate $h \circ h'$ as:

$$\begin{aligned} f \circ T \circ (\text{filter} \circ f') \circ T' \circ \text{filter}' &= f \circ (T \circ g) \circ T' \circ \text{filter}' n \\ &\quad \left\{ \begin{array}{l} g \text{ function with monotone domain,} \\ \text{Lemma 4.6.4} \end{array} \right\} \\ &= (f \circ g') \circ T' \circ \text{filter}' \\ &\quad \left\{ \begin{array}{l} g' : \mathbb{M} \to \mathbb{M} \text{ function with monotone do-} \\ \text{main and codomain, Lemma 4.6.4} \end{array} \right\} \\ &= g'' \circ T' \circ \text{filter}' \\ &\quad \left\{ \begin{array}{l} g'' \text{ function with monotone domain,} \\ \text{Lemma 4.6.4} \end{array} \right\} \end{aligned}$$

This yields an expression of the desired form and by Lemma 4.6.4 each step is in AC^0.

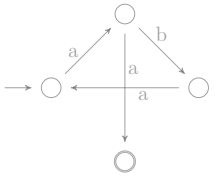
The above calculations depend on the following technical result on the composition of individual functions.

Lemma 4.6.4. *Let $f_{v,k}$ be a filter and $g : \mathbb{M} \to \mathbb{B}^n$ be a function with monotone domain. There are uniform AC^0 circuits calculating $f_{v,k} \circ g$ and $\mathrm{F} \circ g$ and $\mathrm{G} \circ g$, where f's are filters and g is a function with monotone domain.*

Proof. We first show how to compose $g : \mathbb{M} \to \mathbb{B}^n$ and a filter f in AC^0. Let $h = f \circ g$ and for every admissible k let f_k be a Boolean variable which is true if and only if f has offset k. Then for any input $v \in \mathbb{M}$ and index $1 \leq i \leq n$. The value of $h(v)(i)$ equals $(f(g(v)))(i)$. Now, depending on the offset and the pattern p of f, the function $h(v)(i)$ evaluates to true if and only if

$$\bigvee_{k, 1 \leq i+k \leq n} \left\{ f_k \ \wedge \ \left[\begin{array}{l} (p(i+k) = \top) \vee \\ (p(i+k) = \mathrm{ID} \ \wedge \ g(v)(i+k)) \vee \\ (p(i+k) = \mathrm{NOT} \ \wedge \ \neg g(v)(i+k)) \end{array} \right] \right\}$$

is true where $p(i+k) = \top, \mathrm{ID}, \mathrm{NOT}$ is a shorthand for the check that the bits encoding $p(i+k)$ encode the particular pattern. This is a constant-depth polynomial-size formula, thus a constant-depth polynomial-size circuit, as necessary.



We now show how to calculate in AC^0 the compositions $\mathrm{F} \circ g$ and $\mathrm{G} \circ g$ for g a function with monotone domain.

Notice that

$$(\mathrm{F} \circ g)(m_i)(j) = \mathrm{F}(g(m_i))(j) = \bigvee_{k=j\ldots n} g(m_i)(k) = \bigvee_{k=j\ldots n} b_{i,k},$$

which is a constant-depth formula.

Similarly, notice that

$$(\mathrm{G} \circ g)(m_i)(j) = \mathrm{G}(g(m_i))(j) = \bigwedge_{k=j\ldots n} g(m_i)(k) = \bigwedge_{k=j\ldots n} b_{i,k},$$

which is a constant-depth formula. □

In either case, the resulting function is assembled and the composition is in AC^0. Hence, the complexity of the UTL tree-contraction algorithm is $\mathsf{AC}^1[\mathsf{AC}^0] = \mathsf{AC}^1$.

Theorem 4.6.5. UTL *path checking is in* AC^1.

Same results apply to past temporal operators. Note that the construction works also when the negation is applied to arbitrary subformulae, and not only to propositions. Also note that $F_{[a,\infty)}p$ is downward monotone and the corresponding AC^0 circuits are constructible in logarithmic space.

Lemma 4.6.6. *Let* $T = \mathrm{F}_{[a,\infty)}$ *or* $T = \mathrm{G}_{[a,\infty)}$. *Let* $x \in \mathbb{B}^n$ *be a Boolean vector. Then* Tx *is monotone and* $T(x)(i)$ *can be calculated in* AC^0 *for every index* $1 \leq i \leq n$.

Proof. If $T = \mathrm{F}_{[a,\infty)}$ we calculate

$$\mathrm{F}_{[a,\infty)}(x)(i) = \bigvee_{i+a \leq j \leq n} x(i).$$

If $T = \mathrm{G}_{[a,\infty)}$ we calculate

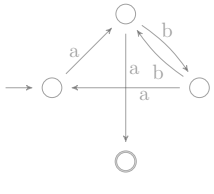
$$\mathrm{G}_{[a,\infty)}(x)(i) = \bigwedge_{i+a \leq j \leq n} x(i).$$

Both expressions are constant-depth circuits and it is easy to see that the output is always monotone. □

Hence we have an AC^1 path-checking algorithm for the more powerful logic $\mathsf{UTL}_{\geq}$ obtained by allowing $F_{[a,\infty)}$ and $G_{[b,\infty)}$ operators.

Lemma 4.6.7. $\mathsf{UTL}_{\geq}$ *path checking is in* AC^1.

To the best of our knowledge, $\mathsf{UTL}_{\geq}$ is the most expressive and powerful fragment of LTL with a sub-$\mathsf{AC}^1[\mathsf{logDCFL}]$ path-checking problem (subject to standard complexity-theoretic assumptions).



	UTL	LTL	MTL	LTL + Xor
Upper Bound	AC^1	AC^1[logDCFL] [KF09]	AC^1[logDCFL]	P
Lower Bound	NC^1 [BCGR92]			P

Table 4.1: Complexity of the path-checking problem.

4.7 Discussion

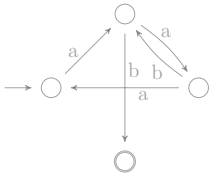
In this chapter, we studied the precise complexity of the path-checking problem for various temporal logics (see Table 4.1). We showed a connection between LTL path checking and planar circuits. It highlights the limitation of the current techniques and provides an evidence to the contrary of the question posed in [KF09] on the existence of a faster algorithm specialised to evaluate circuits arising in the LTL tree contraction algorithm. Further, the connection entails that the complexity of LTL path checking depends on the boolean connectives in general and on addition of non-monotone connectives in particular as the path-checking problem for LTL + Xor is P-complete.

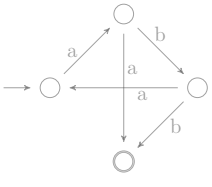
We then showed that the MTL path-checking problem was in AC^1[logDCFL]. This is the most powerful logic we are aware of whose path-checking complexity is in the NC hierarchy. Finally, we presented a more efficient algorithm for the unary fragment UTL of LTL. To the best of our knowledge, this is the most expressive and powerful fragment of LTL with a sub-AC^1[logDCFL] path-checking problem.

Several open problems however remain, the main one being to determine the precise complexity of LTL path checking. In particular, there has been no progress on the trivial NC^1 lower bound.

We brought the complexity of UTL path checking down to AC^1, which is the best possible result using the tree contraction techniques we used. It is conceivable that this could be lowered further using pebble games as in the NC^1 proof of the complexity of formula evaluation [BCGR92].

Note that the reduction to upward-layered circuits makes use of past temporal operators, and an interesting question is whether this is really necessary. Finally, is it possible to separate the complexity of the LTL and MTL path-checking problems?




a
b
a
a
b

Chapter 5

Relationship Between Parametric Timed Automata and Parametric Bounded One-Counter Machines

The problem of reachability in parametric timed automata was introduced over two decades ago in a seminal paper of Alur, Henzinger, and Vardi [AHV93]: given a timed automaton in which some of the constants appearing within guards on transitions are parameters, is there some assignment of integers to the parameters such that an accepting location of the resulting concrete timed automaton becomes reachable?

In this framework, a clock is said to be *nonparametric* if it is never compared with a parameter, and is *parametric* otherwise. Alur *et al.* showed that, for timed automata with a single parametric clock, reachability is decidable (irrespective of the number of nonparametric clocks). The decision procedure given in [AHV93] however has provably nonelementary complexity. In addition, Alur *et al.* showed that reachability becomes undecidable for timed automata with at least three parametric clocks.

The decidability of reachability for parametric timed automata with *two* parametric clocks (and arbitrarily many nonparametric clocks) was left open in [AHV93], with hardly any progress (partial or otherwise) that we are aware of in the intervening

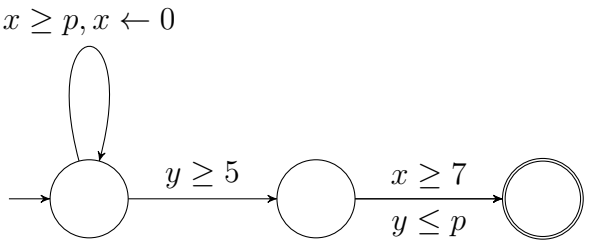
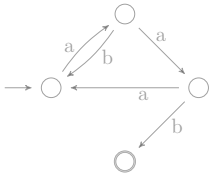


Figure 5.1: An example of a parametric timed automaton. The final state is reachable if, for example, $p = 10$.



period. Alur *et al.* further showed that this problem subsumes the question of reachability in Ibarra *et. al*'s "simple programs" [IJTW93], also open for over 20 years, as well as the decision problem for a fragment of Presburger arithmetic with divisibility.

This and the next chapter are devoted to studying this problem. As is the case in [AHV93], our results are presented for timed automata interpreted over discrete time. However, for parametric timed automata with *closed* (i.e., *non-strict*) clock constraints and parameters restricted to ranging over integers,[1] standard digitisation techniques apply [HMP92, OW03], reducing the reachability problem over dense time to discrete (integer) time.

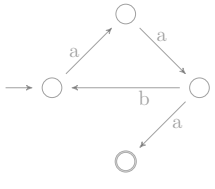
The restriction to integer time enables us, among others, to keep track of the values of two parametric clocks using a single counter, in effect reducing the reachability problem for timed automata with two parametric clocks to the halting problem for parametric bounded one-counter machines. Specifically, in this chapter we show that given a parametric timed automaton with at most two parametric clocks a parametric bounded one-counter machine can be constructed such that the machines have equivalent halting problem.

In case the parametric timed automaton uses only a single parametric clock, the resulting one-counter machine updates the counter only by constants (counter can still be compared to parameters). In the next chapter we then show how to decide the halting problem for this class of one-counter machines.

Further, we show in this chapter that given a parametric bounded one-counter machine a parametric timed automaton with two parametric clocks can be constructed such that the machines have equivalent halting problem. Thus, the decidability of the halting problem for the two classes of automata coincides. The one-counter machine formulation is easier to analyse and in the next chapter we show how to decide the halting problem in certain cases.

All our results do not impose any restriction on the number of nonparametric clocks. It is known [Mil00] that for one parametric clock but no nonparametric clocks the halting problem is NP-complete. Compare this to NEXP-completeness (Theorems 5.2.5 and 5.2.6) if nonparametric clocks are allowed. It is known that reachability in nonparametric timed automata is PSPACE-complete [AD94] and thus allowing nonparametric clocks automatically "extends the automaton with a PSPACE

[1] Other researchers have considered variations in which parameters are allowed to range over rationals, yielding different outcomes as regards the decidability of reachability; see, e.g., [Mil00, Doy07], discussed further below.



machine". This observation is later used in the hardness proofs (Theorem 5.2.6 and Theorem 5.2.11).

5.1 Related Work

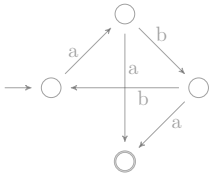
The decidability of reachability for parametric timed automata can be achieved in certain restricted settings, for instance by bounding the allowed range of the parameters [JLR13] or by requiring that parameters only ever appear either as upper or lower bounds, but never as both [HRSV01]: in the latter case, if there is a solution at all then there is one in which parameters are set either to zero or infinity. The primary concern in such restricted settings is usually the development of practical verification tools, and indeed the resulting algorithms tend to have comparatively good complexity. However, solutions to these special cases do not lead to general decision problems as is the case in this thesis.

Miller [Mil00] observed that over dense time and with parameters allowed to range over rational numbers, reachability for parametric timed automata becomes undecidable already with a single parametric clock. In the same setting, Doyen [Doy07] showed undecidability of reachability for two parametric clocks even when using exclusively open (i.e., strict) time constraints.

A connection between timed automata and counter machines was previously established in nonparametric settings [HOW12], and exploited to show that reachability for (ordinary) two-clock timed automata is polynomial-time equivalent to the halting problem for one-counter machines, even when constants are encoded in binary. Unfortunately, it is not obvious how to extend and generalise this construction to parametric timed automata, specifically in the case of two parametric clocks and an arbitrary number of nonparametric clocks, as we handle in the present chapter. The reduction of [HOW12] was used in [FJ13] to show that halting for bounded one-counter machines, and hence reachability for two-clock timed automata, is PSPACE-complete, solving what had been a longstanding open problem.

5.2 Relationship Between Parametric Timed Automata and Parametric Bounded One-Counter Machines

The problem studied in this chapter is the halting problem for parametric timed automata. As stated in the introduction the problem becomes undecidable for automata



with three or more parametric clocks.

In this section we show that given a parametric timed automaton with at most two parametric clocks there is a parametric bounded one-counter machine with equivalent halting problem. Then in the subsequent chapter we show how to decide the halting problem for the latter class.

The convention used in this and the following chapter is that timed automata are denoted by A and counter machines are denoted by C.

As stated in introduction, we restrict to integer-time only. However, assuming that all constraints in the parametric timed automaton A are closed (as is always the case in this thesis), the restriction is unnecessary as a run where clocks can attain arbitrary real-values can be transformed into a run where all clock values are always integers.

Theorem 5.2.1. *Let A be a parametric timed automaton such that all constraints in C are closed (i.e., $c \leq d$, $c \geq d$) then the following are equivalent.*

- *There is an assignment γ_1 such that there is an accepting run π_1 in A^{γ_1} over discrete time.*
- *There is an assignment γ_2 such that there is an accepting run π_2 in A^{γ_2} over dense time.*

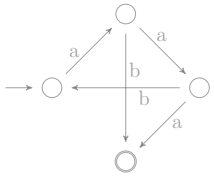
Proof. Clearly, the integer-valued run π_2 is a run in A^{γ_2} also over dense time.

Conversely, suppose that there is such γ_1 and π_1. Then, A^{γ_1} is a closed timed automaton with an accepting run π_1. Hence, by [OW03], there is an integer-valued accepting run π_2 in A^{γ_1}. That is, the clock values in π_2 are always integers. But then π_2 is an accepting run in A^{γ_1} over discrete time. □

5.2.1 Nonparametric Clock Elimination

Let A be a parametric timed automaton. In this section we show, by modifying the region construction [AD94], how to build a parametric timed automaton with equivalent halting problem but without nonparametric clocks.

Note that once the value of a nonparametric clock c is above the largest constant appearing in A, the precise value of the clock does not affect any comparison. Since we restrict to discrete time, the value of c is always a natural number. We eliminate all nonparametric clocks by storing in the state space of A the values of clocks up to the largest constant. However, we must ensure that the eliminated clocks progress simultaneously with the remaining parametric clocks. This motivates parametric 0/1



timed automata where the +1 updates correspond to the progress of time whereas the +0 updates correspond to taking an edge in A.

Definition 5.2.2. *A* parametric 0/1 timed automaton *over the set of clocks C and parameters P is an automaton with*

$$Op = \{+0, +1\} \times 2^C \times G(C, P).$$

Formally, the following result appeared in [AHV93].

Lemma 5.2.3 ([AHV93])**.** *Let $A = (S, s_0, C, P, F, E, \lambda)$ be a parametric timed automaton. Then there is a parametric 0/1 timed automaton $A' = (S', s'_0, C', P', F', E', \lambda')$ such that $C' \subseteq C$ contains only parametrically constrained clocks of C and A and A' have equivalent halting problem. Moreover, $|A'| = O(2^{|A|})$.*

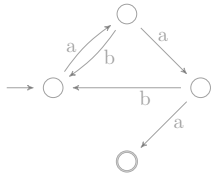
Proof. Let M be the largest constant appearing in A, let S be the set $\{0, 1, \ldots, M + 1\}$ and let X be the set of nonparametric clocks in A. Then the automaton A' is constructed as follows:

- $Q' = Q \times S^X$
- $s'_0 = s_0 \times (0, \ldots, 0)$
- $C' = C \setminus X$
- $P' = P$
- $F' = \{(f, i_1, \ldots, i_{|X|} \mid f \in F \text{ and } i_j \in S\}$
- For a number n let

$$s_M(n) = \begin{cases} M + 1 & \text{if } n = M + 1 \\ n + 1 & \text{otherwise} \end{cases}$$

  and we extend s_M to vectors componentwise.

  The set of transitions E' contains edges of the form $e' = (q \times t, q \times s_M(t))$ with $\lambda'(e') = (+1, \emptyset, true)$ where $t \in S^X$. Further, for a guard G over clocks C and vector $t \in S^X$ assigning a value to each clock in X let $\text{eval}(G, t)$ denotes the expression obtained by instantiating the value of each clock from X by the corresponding value in t. Then if $e = (q, q')$ with $\lambda(e) = (D, G)$ is a transition in E then for every $t \in S^X$ the following transition is in E': $e' = (q \times t, q' \times t')$ with $\lambda'(e') = (+0, D \setminus X, \text{eval}(G, t))$ where $t'(x) = t(x)$ if $x \notin D \setminus X$ and $t'(x) = 0$ if $x \in D \setminus X$. □



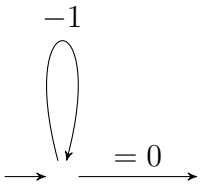


Figure 5.2: Gadget implementing a clock reset.

5.2.2 One Parametric Clock

In case the original parametric timed automaton A uses only one parametric clock we show that the corresponding parametric 0/1 timed automaton has a simple structure.

For the rest of the section, fix a parametric timed automaton A with one parametric clock. By Lemma 5.2.3, there is an exponentially larger parametric 0/1 automaton B with one (parametrically constrained) clock such that A and B have equivalent halting problem.

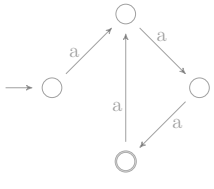
Now, B has a single clock which is incremented by 0 or 1 in each transition. So B is almost a parametric bounded one-counter machine, only that B may contain clock resets. However, clock resets can be eliminated from B by introducing -1 transitions.

Lemma 5.2.4. *Let B be a parametric 0/1 timed one-clock automaton. Then there is a parametric bounded one-counter machine C such that B and C have equivalent halting problem. Further, all updates in C are either $-1, 0$ or $+1$ and $|C| = O(|B|)$.*

Proof. Counter machine C has the same states and $+0, +1$ transitions as B. By modifying B if necessary, we can assume that all edges resetting a clock are $+0$ edges without any comparison (Automata constructed in Lemma 5.2.3 have this property). Each transition resetting a clock is then replaced by a gadget (see Figure 5.2) that subtracts 1 from the counter until the counter equals 0.

Now, each edge of B is labelled by a guard G, which is a conjunction of terms the form $c \leq d, c \geq d$ for a clock c and $d \in \mathbb{N}$. On the other hand, as a minor technical difference, each edge in a one-counter machine can be labelled by at most one comparison. However, by expanding each edge of B into a sequence of edges with each edge labelled by an individual term, we can ensure that every edge in C is labelled by no more than a single comparison. □

Hence, to decide the halting problem for A it suffices to decide the halting problem for a parametric bounded one-counter machine with only $-1, 0, +1$ counter updates. We establish such a result in Theorem 6.1.6, which then yields:



Theorem 5.2.5. *The halting problem for parametric timed automata with one parametric clock is decidable in NEXP time.*

Decidability of the halting problem for the above class of timed automata was first given in [AHV93], albeit with nonelementary complexity. Our proof is different, uses one-counter machines and yields a NEXP algorithm. Moreover, our results are asymptotically optimal. In fact, already for parametric timed automata with a single parameter we have the corresponding lower bound.

Theorem 5.2.6. *The halting problem for parametric timed automata with one parametric clock and one parameter is NEXP-hard.*

Proof. The proof is by reduction from SuccinctSAT, a well-known NEXP-complete problem. A similar construction was used in [GHOW10] to show that LTL model checking of parametric one-counter machines is NEXP-hard.

The SuccinctSAT problem is the adaptation of the classical SAT problem in which the input formula is not given explicitly but instead is encoded by a Boolean circuit $\mathbb{C}$. If $\mathbb{C}$ has k inputs, we require that the size of $\mathbb{C}$ is polynomial in k. Such circuits can encode exponentially larger formulas as follows.

We assume that the input and the output of the circuit $\mathbb{C}$ are bit strings—the values on the input and the output wires. We denote the concatenation of bit strings u and v by $u \cdot v$.

Let b be an n bit number and $i \in \{0, 1, 2\}$. The circuit operates in such a way that evaluating the circuit $\mathbb{C}(b \cdot i) = v \cdot w$ gives the index of the variable $v = f(b, i)$ in the ith literal in the bth clause. Moreover, if the literal is negated then $w = 1$, otherwise, $w = 0$.

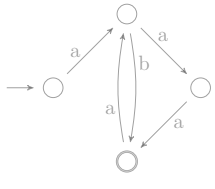
Let φ be a 3SAT formula encoded by circuit $\mathbb{C}$. The variables of φ are denoted by $x_1, x_2, \ldots$. Indexing the clauses of φ by n bit strings, we thus write φ as

$$\varphi = \bigwedge_{b \in \mathbb{B}^n} \Big[(\neg) x_{f(b,0)} \vee (\neg) x_{f(b,1)} \vee (\neg) x_{f(b,2)} \Big].$$

where $f(b, i)$ is the index of the ith variable in the bth clause.

We shall construct a parametric timed automaton A with one parametric clock using a single parameter p such that a final state of A is reachable for some value of p if and only if the value of p encodes a satisfying assignment for φ.

An assignment $v : \{x_1, x_2, \ldots\} \to \{0, 1\}$ for φ is encoded in p as follows. Let π_k be the k-th prime. $v(x_k) = \begin{cases} 1 & \text{if } \pi_k | p \\ 0 & \text{if } \pi_k \nmid p \end{cases}$



Algorithm 5 Algorithm iterating over all clauses of φ and checking that each is satisfiable. $\mathbb{S}_{divides}(p, x)$ stands for subroutine calculating that x divides p and $x \leftarrow \mathbb{S}_{n^{th}prime}(v)$ assigns the nth prime to x.

```
for b ← 0 to 2^n do                                   ▷ Iterate over all clauses
  s ← false                                           ▷ Denotes whether some literal evaluates to true
  for i ← 0 to 2 do                                   ▷ Try all three literals
    v · w ← ℂ(b · i)
    r ← 𝕊_{n^th prime}(v)                             ▷ Calculate the corresponding prime number
    if w = 0 ∧ 𝕊_{divides}(p, r) then
      s ← true
    end if
    if w = 1 ∧ 𝕊_{notdivides}(p, r) then
      s ← true
    end if
  end for
  if s = false then ▷ If the current clause is not satisfied under the assignment; reject
    reject
  end if
end for
accept                                                ▷ All clauses are satisfies; accept
```

That is, $v(x_k) \iff \pi_k | p$. By choosing distinct primes as the basis of the encoding, any assignment to variables in φ can be represented by a number and vice versa.

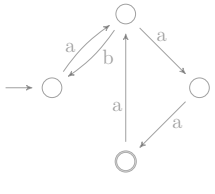
The timed automaton A then implements the algorithm that iterates over all clauses and checks that each clause is satisfiable under the assignment encoded in p (see Algorithm 5). Hence, φ is satisfiable if and only some value of p encodes a satisfying assignment which holds if and only if a final state of A is reachable.

To implement the algorithm using a parametric timed automaton, we provide various gadgets: a gadget to calculate the kth prime number and gadgets to check (non)divisibility of p by the calculated prime numbers.

Note that the problem of calculating the kth prime number is in PSPACE. Now, it is well known that the reachability for nonparametric timed automata is PSPACE-complete [AD94] and so many different ways exist of using timed automata to calculate PSPACE functions. A particular approach is sketched below.

Since A has no restrictions on the number of nonparametric clocks, we use two nonparametric clocks x_a, x_b to store a single bit; interpreting the bit as 1 when $x_a = x_b$ and as 0 otherwise. Note that clock comparisons and resets can be used to check and set the bit, respectively[2]. We use nonparametric clocks as a memory. Now, A uses

[2]Alternatively, resetting the clocks x_a, x_b whenever they reach 1 and checking for the clocks being simultaneously 0 or 1, we can implement bits without direct comparison of two clocks.



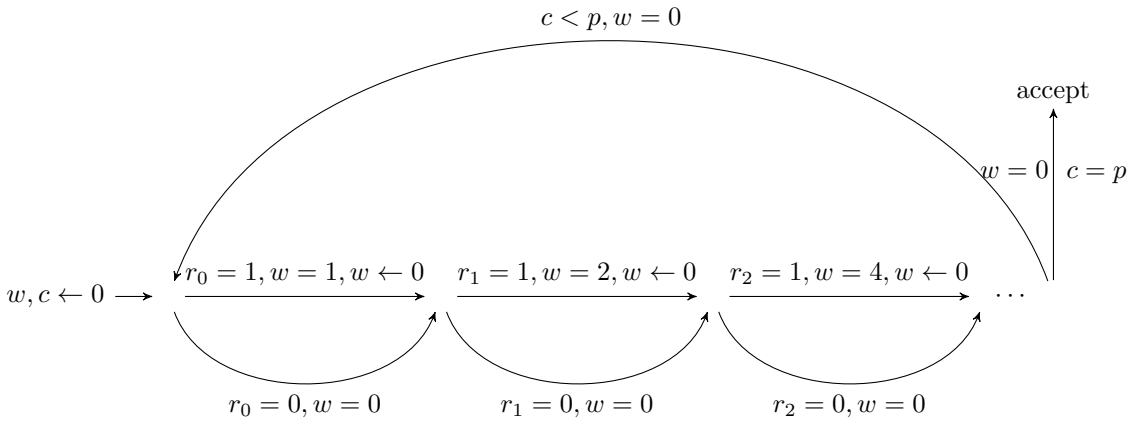


Figure 5.3: Parametric timed automaton implementing r divides p. The bits $r_0, r_1, r_2, \ldots, r_m$ represent $r = \sum_i r_i 2^i$ in binary. An auxiliary clock w is used to count to the powers of 2. The gadget is entered on the left. Note that one traversal through the gadget takes r time units. Clock c measures p. Note that the transition to the accepting state can be taken only if r divides p. Gadget checking $r \nmid p$ is obtained by changing the final test $w = 0, c = p$ to $w = 0, c > p$.

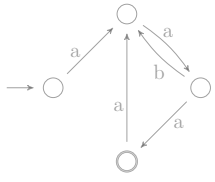
polynomially many clocks to store polynomially many bits. It is easy to see that using the finite state control of A and polynomially many bits, parametric timed automaton A can calculate any PSPACE function. Similarly, we use n bits to iterate over all possible values of b in the algorithm.

In our implementation of the algorithm in Figure 5, we use m bits to store the kth prime number in binary. Further, the algorithm uses subroutines to check whether p is divisible by a given number. We use the gadget shown in Figure 5.3 to check (non)divisibility of p by prime numbers. Notice that the gadgets and hence the entire proof uses only a single parameter p. □

5.2.3 Two Parametric Clocks

We now move to parametric timed automata with two parametric clocks. Decidability in this case is still open and is the main open problem from this thesis. Compare this to reachability in nonparametric timed automata where the precise complexity in the two-clock case was a major open problem until recently shown to be PSPACE-complete [FJ13] (as opposed to NL-complete for one clock [LMS04]). Similarly, for counter machines, the reachability is NP-complete for one counter [HKOW09] but undecidable for two counters [Min67].

In this section we show that the decidability of the halting problem for parametric



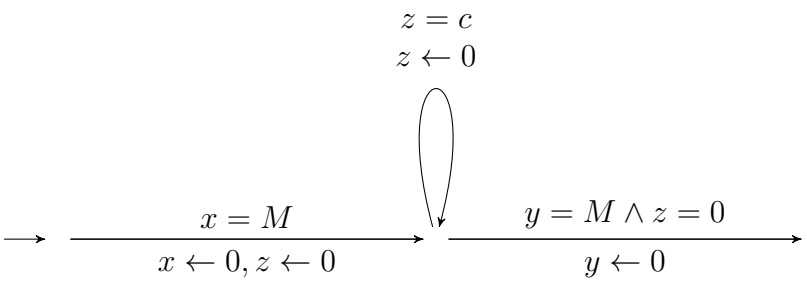


Figure 5.4: Gadget testing that the value stored as $x - y$ is divisible by the given constant $c \in \mathbb{N}$.

timed automata with two parametric clocks is equivalent to the halting problem for parametric bounded one-counter machines. The difference from the results in the previous section is that the counter can be compared as well as incremented/decremented by a parameter.

The equivalence is then used in Section 6.2 to show decidability of the halting problem in case the parametric timed automaton uses only a single parameter.

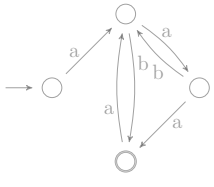
First, observe that a counter can be stored as a difference of two clocks, which can be used to show the easier direction of the equivalence.

Theorem 5.2.7. *Let C be a parametric bounded one-counter machine. Then there is a parametric timed automaton A with two parametric clocks such that A and C have equivalent halting problem. If C has no '$\equiv 0 \bmod c$' transitions then A has no nonparametric clocks. Otherwise, A has one nonparametric clock. Moreover, A is constructible in* P.

Proof. A similar proof appeared in [HOW12]. Timed automaton A tracks the state of C and the counter value of C is stored as the difference $x - y$ of two clocks. Since the clocks progress simultaneously, the difference is constant.

Parametric timed automaton A has the same set of parameters as C together with a fresh parameter M. Intuitively, M is an upper bound on counter values in an accepting run of C. Automaton A ensures that whenever the clock x or y reaches M the clock is reset.

Resetting clocks when they reach M allows us to implement counter operations in A. For example, an update $+p$ can be implemented by resetting y when $y = p$. The test $\leq p$ can be implemented by checking $y = 0 \wedge x \leq p$. And the update '$+[0, p]$' can be implemented by resetting y when $y \leq p$. All other counter updates and comparisons can be implemented similarly.



To check '$\equiv 0 \bmod c$' we introduce a fresh clock z (see Figure 5.4). Clock z resets together with x when $x = M$. Thus, clocks x and z are zero at the same time. Then every time z reaches c, the clock z is reset. Therefore, z counts modulo c. So '$\equiv 0 \bmod c$' reduces to checking that when $y = M$ the value of z equals 0.

Note that since at any single time we make at most one '$\equiv 0 \bmod c$' check, the clock z can be reused in different gadgets for '$\equiv 0 \bmod c$' checks for different constants. Hence only one clock suffices for all '$\equiv 0 \bmod c$' checks.

Finally, observe that x and y are parametrically constrained (by M and parameters already in C) whereas z is not. □

5.2.4 Reduction to Parametric Bounded One-Counter Machines

For the converse, fix A to be a parametric timed automaton with two parametric clocks. We reduce A to a parametric bounded one-counter machine C with equivalent halting problem. As the first step, we construct (Lemma 5.2.3) a parametric 0/1 timed automaton B with two parametrically constrained clocks, denoted x and y, with the halting problem equivalent to A. We now transform B to C.

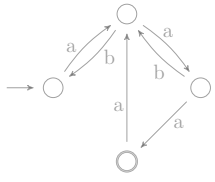
Denote the counter of C by z. For the time being, we need to relax the assumption that z stays nonnegative. That is, subtracting 5 when the counter is 2 results in the counter being -3. In Remark 5.2.9 we later show how to restore the nonnegativity of the counter.

The idea of the reduction is that, after a clock of B is reset, that clock equals zero and we use z to store the value of the other clock. We construct C in such a way that after a reset of y, counter z stores the value of x and after a reset of x, counter z stores $-y$. Initially C starts with the counter equal to 0.

Machine C then operates in phases. Each phase corresponds to a run of B between two consecutive resets of some (possibly different) clock.

Suppose y was the last clock to reset. After the reset, the configuration of B is $(s, (z, 0))$ for some state $s \in B$ and the counter $z = x$. We show how C calculates the configuration after the next clock reset in B.

After time Δ, the clocks go from configuration $(z, 0)$ to $(z + \Delta, \Delta)$. Based on the guards, different transitions in B^γ are enabled as time progresses. For the time being, we assume that the order of the parameters is $p_1 < p_2 < \ldots < p_k$. Later we relax this assumption by building a separate automaton for each possible permutation.



Let *region* $R_{(i,j)}$ be the set of clock valuations $[p_i, p_{i+1}] \times [p_j, p_{j+1}]$. Then the set of enabled transitions depends only on the region $R_{(i,j)}$ the clocks (x, y) lie in.[3]

Therefore, machine C guesses (see Figure 5.5):

- the regions $R_{(i_0,j_0)}, R_{(i_1,j_1)}, \ldots, R_{(i_m,j_m)}$ in the order in which they are visited by the clocks (x, y),
- the states $s_0, s_1, \ldots, s_m$ of B when each region R_l is visited for the first time,
- the state t of B in which the next reset occurs,
- which clock is reset next.

Machine C then checks that the sequence is valid. First, C checks, that $(z, 0)$ lies in R_0. Second, it checks that the regions are adjacent:

$$(i_{l+1} - i_l = 1 \wedge j_{l+1} = j_l) \text{ or } (i_{l+1} = i_l \wedge j_{l+1} - j_l = 1) \text{ or } (i_{l+1} - i_l = j_{l+1} - j_l = 1).$$

The last case corresponds to the clocks hitting a corner of a region. Note that this forces $m \leq 2k$. Then, C checks that starting in clock configuration $(z, 0)$, the regions can be visited in the guessed order.

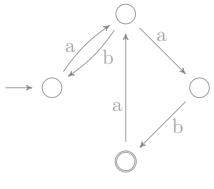
Consider region $R_{(u,v)}$ for some u, v. When the region is visited for the first time, then either clock x equals p_u or clock y equals p_v. In the former case, the clock configuration of B is $(p_u, p_u - z)$, in the latter case, it is $(p_v + z, p_v)$. The configuration depends on the direction in which $R_{(u,v)}$ is visited. See Figure 5.5.

- If $i_{l+1} - i_l = 1$ then C checks that clock x reaches $p_{i_{l+1}}$ before clock y reaches p_{j_l+1}. That is, $p_{i_{l+1}} - z \leq p_{j_l+1}$. Equivalently, $p_{i_{l+1}} \leq z + p_{j_l+1}$, which can be easily tested by a parametric bounded one-counter machine[4]. In Figure 5.5 this corresponds to region $R_{(1,0)}$, which is visited before $R_{(2,0)}$.
- Similarly, if $j_{l+1} - j_l = 1$ then C checks that clock y reaches $p_{j_{l+1}}$ before clock x reaches p_{i_l+1}. That is, $p_{i_l+1} - z \geq p_{j_{l+1}} - z$. Equivalently, $p_{i_l+1} \geq p_{j_{l+1}} + z$, which can be easily tested by a parametric bounded one-counter machine. In Figure 5.5 region $R_{(2,1)}$ is visited before $R_{(2,2)}$.

We say that R_l was *reached from left* in the first and that R_l was *reached from bottom* in the second case. See Figure 5.5 for the intuition behind the names.

[3]Our definition of rectangular regions differs slightly from the one usually given in the literature [AD94]. However, as all inequalities are nonstrict the regions are sufficient. For ease of presentation, we also use the convention $p_0 = 0$ and $p_{k+1} = \infty$.

[4]The test can be implemented by the sequence of the following edges: $+p_{j_l+1}, \geq p_{i_{l+1}}, -p_{j_l+1}$



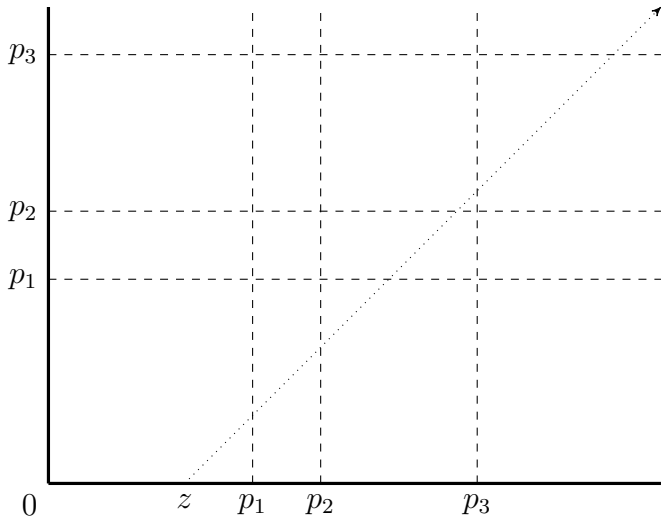


Figure 5.5: Regions for three parameters $p_1 < p_2 < p_3$. The dotted line shows an evolution of clock configuration, which visits $R_{(0,0)}, R_{(1,0)}, R_{(2,0)}, R_{(2,1)}, R_{(2,2)}, R_{(3,2)}, R_{(3,3)}$.

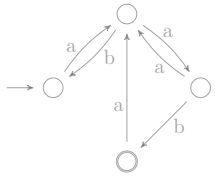
Finally, C checks reachability within individual regions. Let $\boldsymbol{c_l}$ be the configuration in which the region R_l is visited for the first time. Then C checks that a run from $\boldsymbol{c_l}$ to $\boldsymbol{c_{l+1}}$ exists in R_l.

Now, with each $R_{(i,j)}$, we introduce a one-counter machine $B_{(i,j)}$ obtained from B assuming clock $x \in [p_i, p_{i+1}]$ and clock $y \in [p_j, p_{j+1}]$, instantiating all comparisons accordingly and by removing all edges resetting a clock.

Formally, for each i and j, the automaton $B_{(i,j)}$ is obtained from B by

- removing all edges resetting a clock,
- removing all edges labelled by $x \geq p_k$ for $k > i$,
- removing all edges labelled by $x \leq p_k$ for $k \leq i$,
- replacing by an $+0$ edge all edges labelled by $x \geq p_k$ for $k \leq i$,
- replacing by an $+0$ edge all edges labelled by $x \leq p_k$ for $k > i$,
- removing all edges labelled by $y \geq p_k$ for $k > j$,
- removing all edges labelled by $y \leq p_k$ for $k \leq j$,
- replacing by an $+0$ edge all edges labelled by $y \geq p_k$ for $k \leq j$,
- replacing by an $+0$ edge all edges labelled by $y \leq p_k$ for $k > j$,

Notice that $B_{(i,j)}$'s are 0/1 automata without resets, comparisons or parameters, i.e, one-counter machines. In particular, the reachability relation for $B_{(i,j)}$'s is semilinear. Formally, for a pair of states s and t of a one-counter machine X define $\Pi(X, s, t)$



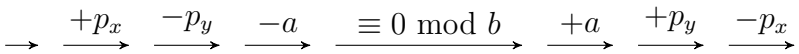


Figure 5.6: Gadget testing that for given $a, b \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that $z+p_x-p_y = a + kb$, i.e., $z + p_x - p_y - a \equiv 0 \bmod b$. Letter z denotes the current counter value.

to be the set of counter values reachable at t by a run starting in state s and counter equal to 0: $\Pi(X, s, t) = \{v \mid \exists \pi \in X \,.\, \mathrm{start}(\pi) = (s, 0) \ \wedge \ \mathrm{last}(\pi) = (t, v)\}$.

Lemma 5.2.8. *Let X be a one-counter machine with 0/1 updates. Then for any states $s, t \in X$ the set $\Pi(s, t)$ is effectively semilinear: $\Pi(X, s, t) = D \cup \bigcup_{1 \le j \le r} \{a_j + b_j\mathbb{N}\}$ where $D \subseteq \mathbb{N}$ is finite and $a_j, b_j \in \mathbb{N}$.*

Proof. It was shown in [Haa12], Lemma 4.1.18, that the reachability relation in non-parametric one-counter machines is definable in the existential fragment of Presburger arithmetic (without divisibility). It is well known that this fragment defines effectively semilinear sets. □

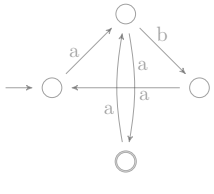
Edges in $B_{(i,j)}$ never decrease the clock and hence the restriction to paths starting with the counter equal to 0 in the definition of $\Pi(X, s, t)$ is sufficient.

Now, to check that a run from $\boldsymbol{c_l}$ to $\boldsymbol{c_{l+1}}$ exists in R_l, machine C distinguishes whether R_l and R_{l+1} are reached from bottom or from left and uses the semilinearity of the reachability relation of the corresponding $B_{(i,j)}$.

The translation is technical. For example, suppose $R_l = R_{(p_x,p_y)}$ for some parameters p_x and p_y. Then $\boldsymbol{c_l} = (s_l, (p_x, p_x - z))$ or $\boldsymbol{c_l} = (s_l, (p_y + z, p_y))$ depending on the direction. If R_l was reached from left and R_{l+1} from bottom then C has to check that $(s_{l+1}, (p_{y+1} + z, p_{y+1}))$ is reachable from $(s_l, (p_x, p_x - z))$. That is, that $z+p_{y+1}-p_x \in \Pi(B_l, s_l, s_{l+1})$. This and all other semilinear constraints can be checked by C using '$\equiv 0 \bmod c$' transitions (cf. Figure 5.6).

We now explain how to handle the remaining cases. Consider region $R_l = R_{(p_x,p_y)}$ for some parameters p_x and p_y, i.e., $R_l = (p_x, p_{x+1}) \times (p_y, p_{y+1})$. Then, R_l is visited for the first time in configuration $\boldsymbol{c_l} = (s_l, (p_x, p_x - z))$ or $\boldsymbol{c_l} = (s_l, (p_y + z, p_y))$. Then C checks that it is possible to go from $\boldsymbol{c_l}$ to $\boldsymbol{c_{l+1}}$ in R_l. The check distinguishes whether R_l and R_{l+1} were reached from bottom or from left.

- R_l **reached from left,** R_{l+1} **reached from left**. Thus, C has to check that $(s_{l+1}, (p_{x+1}, p_{x+1} - z))$ is reachable from $(s_l, (p_x, p_x - z))$. That is, that $p_{x+1}-p_x \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C (similarly as in Figure 5.6).



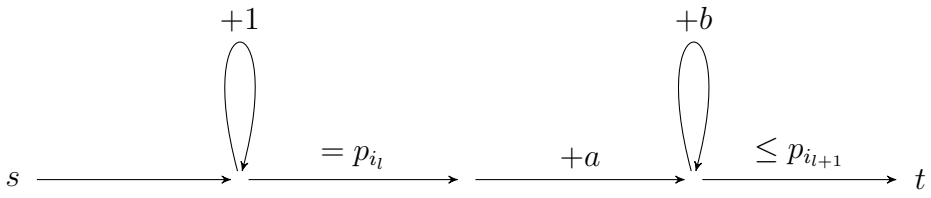


Figure 5.7: A gadget implementing a reset of clock y in the left/left situation.

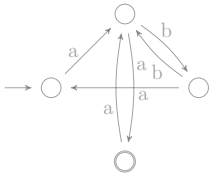
- R_l **reached from left,** R_{l+1} **reached from bottom**. Thus, C has to check that $(s_{l+1}, (p_{y+1} + z, p_{y+1}))$ is reachable from $(s_l, (p_x, p_x - z))$. That is, that $p_{y+1} + z - p_x \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C (Figure 5.6).
- R_l **reached from bottom,** R_{l+1} **reached from left**. Thus, C has to check that $(s_{l+1}, (p_{x+1}, p_{x+1} - z))$ is reachable from $(s_l, (p_y + z, p_y))$. That is, that $p_{x+1} - p_y - z \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C.
- R_l **reached from bottom,** R_{l+1} **reached from bottom**. Thus, C has to check that $(s_{l+1}, (p_{y+1} + z, p_{y+1}))$ is reachable from $(s_l, (p_y + z, p_y))$. That is, that $p_{y+1} + z - p_y - z = p_{y+1} - p_y \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C.

Finally, we show how to simulate by C a clock reset in B. Suppose that instead of resetting the clock, we let the time evolve. Then eventually the clock configuration transitions to another region. Denote that region by R_{l+1}.

First, suppose that y is the next clock to be reset. Then the first thing to check is whether there is a transition leaving t that resets the clock y.

Simulation of a clock reset distinguishes whether R_l and R_{l+1} were reached from bottom or from left. By Lemma 5.2.8, the set $\Pi(B_l, s_l, t)$ is semilinear. So, C guesses a generator (a, b) in $\Pi(B_l, s_l, t)$ that it will use to reach t. Then C checks whether state t can be reached using that generator. There are four cases to consider depending on the direction from which R_l and R_{l+1} are reached. In all four cases, automaton C nondeterministically chooses the counter value when t is reached.

- R_l **reached from left,** R_{l+1} **reached from left**. Thus, B reaches R_l in the configuration $(s_l, (p_x, p_x - z))$. Since R_{l+1} would be reached from left, machine B resets when clock $x < p_{x+1}$. So C has to set the counter to a value $p_x + \Pi(B_l, s_l, t) \in [p_x, p_{x+1}]$. To do that, C first sets the counter to p_x. Then C adds a to the counter and starts nondeterministically incrementing the counter by b, checking that the counter stays below p_{x+1}. See Figure 5.7.



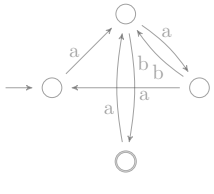
- R_l **reached from left,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in the configuration $(s_l, (p_x, p_x - z))$. And B resets when clock $x < p_{y+1} + z$. Thus, the new counter value $z' = p_x + a + kb \leq z + p_{y+1}$. So C increments the counter by p_{y+1}. Then, it keeps subtracting 1 from the counter checking that it is at least $\geq p_x + a$. Then C nondeterministically terminates when it holds that $z' - p_x - a \equiv 0 \bmod b$.
- R_l **reached from bottom,** R_{l+1} **reached from left**. Thus, B reaches R_l in the configuration $(s_l, (p_y + z, p_y))$. And B resets when clock $x < p_{x+1}$. Now, C first increments the counter to p_y. Second, it adds a to the counter and starts nondeterministically incrementing the counter by b, checking that the counter stays below p_{x+1}.
- R_l **reached from bottom,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in the configuration $(s_l, (p_y + z, p_y))$. And B resets when clock $x < p_{y+1} + z$. Thus C increments the counter by a number in the range $p_y + [0, p_{y+1} - p_y]$ of the form $a + kb$. This can be achieved, by first calculating $z \mod b$ using the '$\equiv 0 \bmod b$' transitions, then incrementing the counter using $+[0, p_{y-1} - p_y]$ and then checking that the new counter value z' satisfies $z' \equiv z + a \bmod b$.

This finishes a reset of y. For the reset of x, we proceed analogously. Consider the four cases:

- R_l **reached from left,** R_{l+1} **reached from left**. Thus, B reaches R_l in configuration $(s_l, (p_x, p_x - z))$ and resets when clock $y < p_y - z$. Recall that after reset, the new counter value z' stores $-y$. Thus, $z' \in [z - p_{x+1}, z - p_x]$. Analogously to (bottom, bottom) case above, B subtracts p_x from the counter and then B subtracts a number of the form $a + kb \in [0, p_{x+1} - p_x]$.
- R_l **reached from left,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in configuration $(s_l, (p_x, p_x - z))$ and resets when $y < p_{y+1}$.

  So, $z' \in [-p_{y+1}, z - p_x]$. So B subtracts p_y and then a from the counter and then B keeps subtracting b checking that the counter is above $-p_{x+1}$.
- R_l **reached from bottom,** R_{l+1} **reached from left**. Thus, B reaches R_l in configuration $(s_l, (p_y + z, p_y))$ and resets when clock $y < p_{x+1} - z$.

  So, $z' \in [z - p_{x+1}, -p_y]$. Hence, B subtracts p_{x+1} and then adds a to the counter and then B keeps adding b checking that the counter is below $-p_y$.



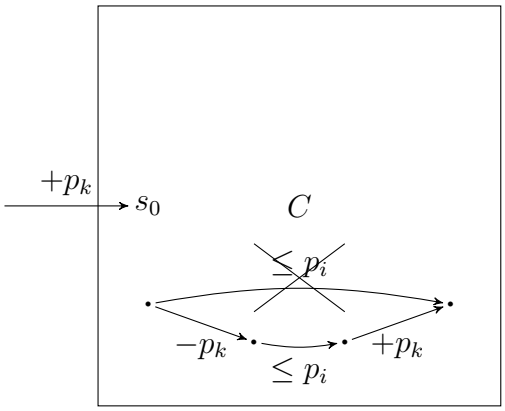


Figure 5.8: Automaton C' obtained from C by offsetting the counter by p_k. Each comparison edge is replaced by three edges which subtract p_k from the counter, perform the original check and then add p_k to the counter.

- R_l **reached from bottom,** R_{l+1} **reached from bottom** Thus, B reaches R_l in configuration $(s_l, (p_y+z, p_y))$ and resets when $y < p_{y+1}$. So, $z' \in [-p_{y+1}, -p_y]$. So B sets the counter to $p_y - a$ and then it keeps subtracting b checking that the counter is above $-p_{y+1}$.

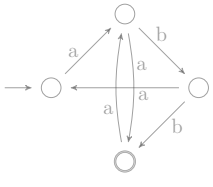
This finishes the simulation of a single stage of B. Now, notice that once the value of a clock becomes larger than p_k its exact value is irrelevant to any future comparison. Hence, C tracks x and y only up to p_k and remembers which clocks exceed it. Hence, we can assume that the counter of C is always inside $[-p_k, p_k]$.

Next, we modify C to ensure that the counter is always nonnegative (and inside $[0, 2p_k]$). See Figure 5.8 for the automaton we build. Let C' be obtained from C by adding a new initial state and a $+p_k$ edge from the new to the original initial state. Further, any comparison edge (s, G, t) (e.g., where G is $\leq p_i$) is replaced by a gadget of three edges $(s, -p_k, q), (q, G, q')$ and $(q', +p_k, t)$ which subtract p_k from the counter, perform the original check and then add p_k to the counter thereby offsetting the counter by p_k.

Remark 5.2.9. *We can assume that the counter of C is always inside $[0, 2p_k]$.*

Note that the construction depends on the order of parameters. However, we can build an automaton for every possible order of parameters. Then check the order of parameters and transition into the automaton for the appropriate order (see Figure 5.9).

Finally, note that all '$\equiv 0 \bmod c$' transitions are nonparametric. That is, $c \in \mathbb{N}$ is always a natural number and never a parameter.



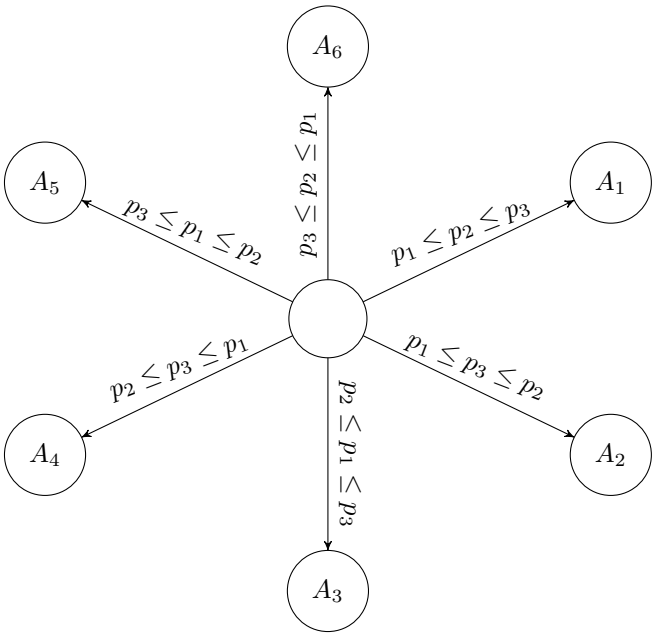


Figure 5.9: Parametric bounded one-counter automaton checking the order of parameters and transitioning into a corresponding subautomaton.

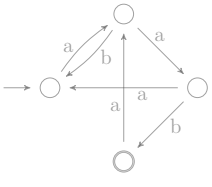
Theorem 5.2.10. *For a parametric timed automaton A with two parametric clocks we can compute a parametric bounded one-counter machine C with equivalent halting problem.*

Our reduction was inspired by the work in [HOW12] (see the Related Work section). Performing one phase in a single stage of C and using semilinearity of reachability in individual regions are the main differences from the reduction of [HOW12].

Already for a single parameter, we have the following lower bound. Subject to the standard complexity-theoretic assumptions, the lower bound is much higher than the upper bound for the case of one parametric clock (NEXP, Theorem 5.2.5). However, note that $\mathsf{PSPACE}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$ [AKRR11].

Theorem 5.2.11. *The halting problem for parametric timed automata with two parametric clocks and a single parameter is* $\mathsf{PSPACE}^{\mathsf{NEXP}}$*-hard.*

Proof. We first describe the $\mathsf{PSPACE}^{\mathrm{NEXP}}$-hard problem we reduce from. Recall from the proof of Lemma 5.2.6 that SuccinctSAT, whereby a 3SAT formula is represented by a Boolean circuit, is a NEXP-complete problem. So consider a PSPACE Turing Machine T that makes NEXP oracle calls by means of SuccinctSAT. Precisely, T is allowed to generate polynomial-size circuits encoding 3SAT formulae and then pass



them to a SuccinctSAT oracle. It is clear that such machines can solve $\mathsf{PSPACE}^{\mathsf{NEXP}}$ problems.

Given such a machine T, we now describe how to build in polynomial time a parametric timed automaton with two parametric clocks simulating T. As the first step, we build a polynomial-size parametric timed automaton A with two parametric clocks deciding SuccinctSAT. This automaton is then used inside a nonparametric timed automaton as a NEXP oracle. Hence, the $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hardness follows.

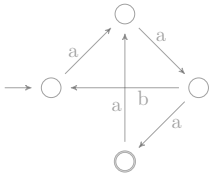
The construction of A is an extension of ideas from Lemma 5.2.6 for building a parametric timed automaton with one parametric clock. Recall that in the lemma we used a circuit $\mathbb{C}$ to encode a 3SAT formula φ. Further recall that in the lemma, we described how to use nonparametric clocks as memory. Automaton A uses polynomially many clocks to represent polynomial-size memory. The input to A is the description of circuit $\mathbb{C}$ encoded in this polynomial-size memory.

If $\mathbb{C}$ has $n+2$ inputs then φ can have as many as $3 \cdot 2^n$ variables, which we denoted by $x_1, x_2, \ldots$. Further recall that we represented an assignment v to variables in φ by a number $Z \in \mathbb{N}$ such that $v(x_k) = 1$ if and only if $\pi_k | Z$ where π_k is the kth prime.

Now, $\pi_k \leq 2k^2$. Hence, any valid assignment is represented by a number at most $lcm(1, 2, 3, \ldots, 2(3 \cdot 2^n)^2)$. On the other hand, $\pi_k \geq k$ and so some assignments are represented only by numbers greater than $(3 \cdot 2^n)!$, which is doubly exponential in n. Thus, storing Z directly would require exponentially many bits.

Let M be a parameter and suppose that $M \geq lcm(1, 2, 3, \ldots, 2(3 \cdot 2^n)^2)$. Then instead of using exponentially many bits, automaton A stores the value of an assignment as a difference $x - y$ of two parametric clocks x and y. The clocks operate modulo M. That is, whenever a clock reaches value M, the clock is reset. With this convention, the assignment stored by clocks x and y is the value of x when y equal 0. Notice that with this convention, resetting y when $y = 1$ increments the value of the stored assignment by one. The resetting trick is used by A to iterate over all assignments. The initial values of x and y, and hence the assignment, is 0.

Automaton A then checks whether an individual assignment satisfies φ similarly as is done in Lemma 5.2.6. Automaton A implements the Algorithm 6 that iterates over all clauses and checks that each is satisfied by the assignment. In order to check that an individual assignment satisfies φ, automaton A needs to be able to extract the value of each variable from the assignment. That is, A needs to be able to calculate the value of the kth prime π_k and to check whether π_k divides the value of the assignment. In Lemma 5.2.6 we described how A can calculate and represent π_k in binary $\pi_k = \sum_i r_i 2^i$ where r_i are bits.



Then checking whether π_k divides the assignment is done by modifying the gadget from Figure 5.3 used for the divisibility in Lemma 5.2.6. We modify the gadget so that it can be entered only when x is reset and it can be exited only when y is reset. This guarantees that exactly $x-y$ timeunits elapse during the traversal of the gadget.

This way, timed automaton A can iterate over all assignments for φ and check if at least one makes the formula true. Finally, to ensure that M is big enough, the automaton iterates over all numbers $k \in \{1, 2, 3, \ldots, 2(3 \cdot 2^n)^2\}$ and checks that $k|M$. This is possible, without using any additional parameters, as only polynomially many bits (clocks) are needed to store the value of each possible $k = 1, \ldots, 2(3 \cdot 2^n)^2$. The algorithm that is hard-wired into A is described in Algorithm 6. Notice that A uses only two parametric clocks (x and y). Further notice that the only input to the algorithm is the circuit $\mathbb{C}$. The function f which takes an encoding of a circuit $\mathbb{C}$, input x and returns $f(\mathbb{C}, x) = \mathbb{C}(x)$ the value of $\mathbb{C}$ on x, is PSPACE computable. So we can modify the algorithm in Figure 6 to take an encoding of a circuit and accept if and only if the corresponding formula is satisfiable. Since SuccinctSAT is NEXP-complete, the algorithm can be used as a general NEXP oracle. The resulting automaton is constructible in polynomial time and is of polynomial size in the input. Thus, we can use it as an oracle in a PSPACE algorithm.

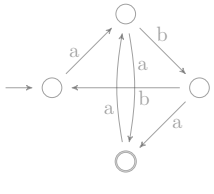
So the parametric timed automaton B simulating T works as follows. Automaton B uses polynomially many bits (clocks) to simulate T, making the same transitions as T does. Then whenever T makes an oracle call, automaton B resets x and y and then starts executing parametric timed automaton A on the circuit encoding the corresponding 3SAT formula.

Hence, there is a reduction from $\mathsf{PSPACE}^{\mathsf{NEXP}}$ problems to the halting problem for parametric timed automata with two parametric clocks. Observe that the value of the parameter M can be reused between the "oracle" calls, as M is an upper bound on the largest valid assignment encoded by a circuit with n inputs. □

5.3 Discussion

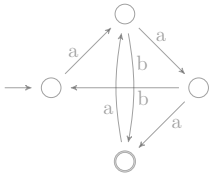
In this Chapter we gave a reduction from the halting problem for parametric timed automata with at most two parametric clocks to the halting problem for parametric bounded one-counter machines.

The halting problem for timed automata with three or more parametric clocks is known to be undecidable whereas the two-parametric-clock case is a long-standing open problem [AHV93]. In the following chapter we give decidability results for



Algorithm 6 Algorithm iterating over all valid assignments and for each assignment iterating over all clauses of φ and checking that each is satisfiable. The automaton accepts if the formula is satisfiable and rejects otherwise.

for $i \leftarrow 1$ **to** $2(3 \cdot 2^n)^2$ **do**
  if $\mathbb{S}_{notdivides}(M, i)$ **then** ▷ Checks that $lcm(1, \ldots, 2(3 \cdot 2^n)^2) | p$
    reject
  end if
end for
$ass \leftarrow 0$ ▷ Stores the value of the assignment
while $ass \neq M$ **do** ▷ Try all possible assignments
  for $b \leftarrow 0$ **to** 2^n **do** ▷ Iterate over all clauses
    $ok \leftarrow$ true ▷ Denotes whether the current assignment satisfies φ
    $s \leftarrow$ false
    for $i \leftarrow 0$ **to** 2 **do** ▷ Check if some literal is true under the current assignment
      $v \cdot w \leftarrow \mathbb{C}(b \cdot i)$
      $r \leftarrow \mathbb{S}_{n^{th}prime}(v)$
      if $w = 0 \ \wedge \ \mathbb{S}_{divides}(M, r)$ **then**
        $s \leftarrow$ true
      end if
      if $w = 1 \ \wedge \ \mathbb{S}_{notdivides}(M, r)$ **then**
        $s \leftarrow$ true
      end if
    end for
    if $s =$ false **then**
      $ok \leftarrow$ false
    end if
  end for
  if $ok =$ true **then** ▷ Accept if found a satisfying assignment
    accept
  end if
end while
reject ▷ No assignment satisfies φ; reject



certain classes of parametric bounded one-counter machines. In particular, we match the NEXP lower-bound for one parametric clock.

Even though we were unable to prove the case of two parametric clocks in full generality in this thesis, we believe that the reduction to one-counter machines might play an important step in resolving the problem as was the case in nonparametric settings where a similar reduction [HOW12] led to the resolution of precise complexity for the reachability problem in two-clock timed automata [FJ13].

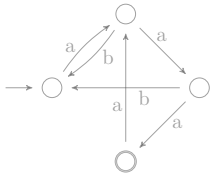
The reduction from automata with one parametric clock produces exponentially larger one-counter machines, which in presence of the NEXP lower bound (Theorem 5.2.6), is the best achievable.

For parametric timed automata with two parametric clocks, however, the situation is more complicated. We established a $\mathsf{PSPACE}^{\mathsf{NEXP}}$ lower bound. It is conceivable that the lower bound can be further improved to EXPSPACE. In proof of the lower bound (Theorem 5.2.11), we employed polynomially many nonparametric clocks to store polynomially many bits. Hence polynomially many clocks can encode numbers exponentially large in magnitude, which could then be used to point into exponential memory. However, we were unable to devise a way of storing exponential many bits in two parametric clocks which would then yield an EXPSPACE lower bound.

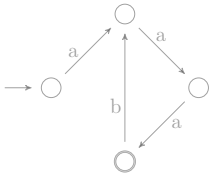
On the other hand, for a parametric timed automaton A with two parametric clocks, the corresponding parametric bounded one-counter machine C is constructed via a sequence of operations each of which is exponential in either the number of states, number of nonparametric clocks or the number of parameters. First, a 0/1 timed automaton B is constructed (incurring exponential blow-up in the number of nonparametric clocks). Then the order of regions is guessed (incurring exponential blow-up in the number of parameters), then semilinearity checks are performed (incurring exponential blow-up in the size of B). Thus, the resulting one-counter machine C is doubly exponential in the number of nonparametric clocks and exponential in the number of states and parameters in A.

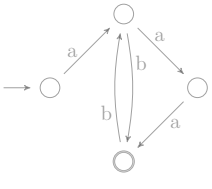
It is left as an open problem whether a more efficient reduction is possible. However, as the decidability of the problem is not know, such a result is of much lesser importance.

Finally, note that the reduction from counter machines to timed automata introduces an extra parameter whereas the opposite reduction does not. Thus, decidability in the k-parameter fragment for parametric bounded one-counter machines is harder



than for the k-parameter fragment for parametric timed automata with two parametric clocks. In the next section we give decision procedures for the one-parameter fragment in both classes.




a
a
b
b
a

Chapter 6

Decidability Results for Parametric Bounded One-Counter Machines

We have shown in the previous section that the decidability of the halting problem for parametric timed automata with two parametric clocks is equivalent to the halting problem for parametric bounded one-counter machines. The decidability of the former problem was left open in [AHV93], with hardly any progress (partial or otherwise) in the period since its introduction. The latter class of automata is easier to reason about and we study the decidability of the latter problem in this chapter. We further show in later sections how the decidability results generalise to the reachability in simple programs of O. Ibarra *et al.* [IJTW93]—another long-standing open problem.

The resolution of the halting problem for general parametric bounded one-counter machines is the main open question of the thesis. However, motivated by the special classes arising in the reduction from parametric timed automata, we prove decidability of the halting problem in some cases.

In particular, we derive a NEXP algorithm for parametric timed automata with a single parametric clock, a decision procedure for parametric timed automata with two parametric clocks and a single parameter, and a decision procedure for parametric

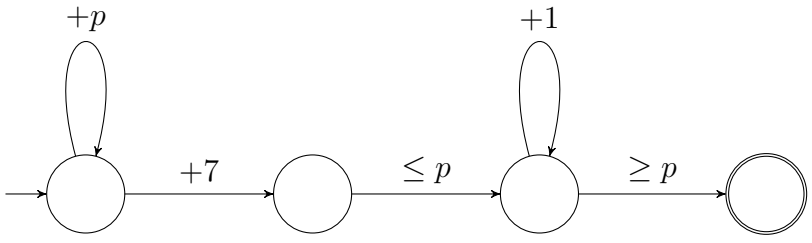
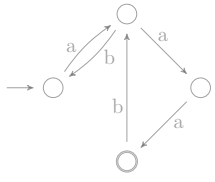


Figure 6.1: An example of a parametric bounded one-counter machine. The final state is reachable for all $p \in [7, \infty)$.



bounded one-counter machines with a single parameter.

Our results rest in part on new developments in the theory of one-counter machines [HKOW09] and their encodings in Presburger arithmetic [Haa12]. As no upper bounds are imposed on the value of the counter in the above publications and as their techniques make crucial use of the unboundedness of the counter, their results are, however, not directly applicable in the present setting.

6.1 Machines with Constant Updates

In Section 5.2.2 we showed that the halting problem for parametric timed automata with one parametric clock reduces to the halting problem for parametric bounded one-counter machines with all counter updates either $-1, 0$ or $+1$. For the rest of the section, fix a machine C of the latter type. We now show how to decide the halting problem for C.

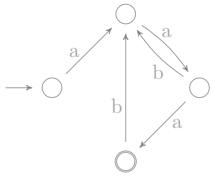
To show that C halts, we have to find an assignment γ and an accepting run π in C^γ. Even without knowing γ, we show that π splits into subruns of a simple form independent of γ the existence of which is reducible to satisfiability of certain $\exists$PAD formulae.

Let γ be a parameter assignment and assume that we guessed the order of parameters, let's say, $\gamma(p_1) < \gamma(p_2) < \ldots < \gamma(p_k)$, but not the precise values of parameters. Let $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ be arbitrary configurations of C^γ such that $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ in C^γ.

We now show how to decide the existence of a run $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ in C^γ. Consider a shortest run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$. There is a constant $M \in \mathbb{N}$, determined in Lemma 6.1.3, such that the run π can be factored into subruns between successive parameters and subruns around individual parameters (see Figure 6.2). Formally, $\pi = \pi_0 \to \pi_1 \to \pi_2 \to \cdots \to \pi_l$ such that (π_0 can be possible empty)

- Even-indexed runs: $\gamma(p_r) - M \leq \text{counter}(\pi_{2i}) \leq \gamma(p_r) + M$ for some parameter p_r,
- Odd-indexed runs: $\gamma(p_r) + M < \text{counter}(\pi_{2i+1}) < \gamma(p_{r+1}) - M$ for some consecutive parameters $\gamma(p_r) < \gamma(p_{r+1})$,
- Even- and odd-indexed runs are joined by an edge: $\text{last}(\pi_i) \to \text{start}(\pi_{i+1})$.

Now, notice that every transition in C changes the counter by at most 1. Hence, $\text{counter}(\text{start}(\pi_{2i})) = p_r + M$ or $\text{counter}(\text{start}(\pi_{2i})) = p_r - M$ for some parameter p_r. Similarly, $\text{counter}(\text{start}(\pi_{2i+1})) = p_r + M + 1$ or $\text{counter}(\text{start}(\pi_{2i+1})) = p_{r+1} - M - 1$. Thus, $\text{start}(\pi_i)$ is always of the form $\text{start}(\pi_i) = (s_i, p_{f(i)} + x)$ for some state s_i,



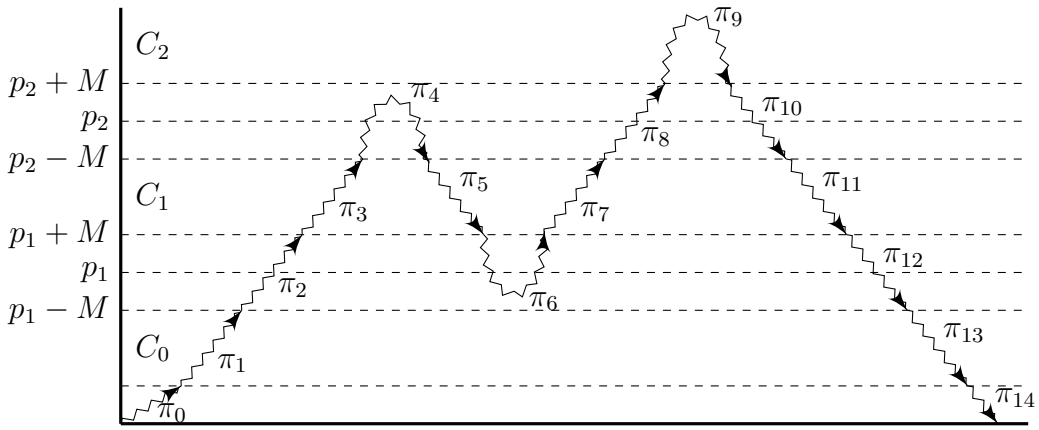


Figure 6.2: Factoring of a run, which starts and finishes with the counter equal to 0. The y axis shows the counter value, hypothetical values of the parameters and their neighbourhoods. Label C_i marks the interval corresponding to counter machine C_i.

some $|x_i| \in \{M, M+1\}$ and parameter $p_{f(i)}$. Hence, $\text{start}(\pi_i)$ is uniquely determined by the triple $(s_i, f(i), x_i)$. Similarly, $\text{last}(\pi_i)$ is uniquely determined by some triple $(t_i, g(i), y_i)$ with $|y_i| \in \{M, M+1\}$.

By minimality, π visits every configuration at most once. Hence an odd-indexed run can start in only one of $2nk$ configurations (n states, k parameters). Hence, the number of odd-indexed runs, and hence the total number of runs is $O(nk)$.

Lemma 6.1.1. *Let π be a shortest accepting run in C^γ. Then the factoring of a π has $O(nK)$ parts.*

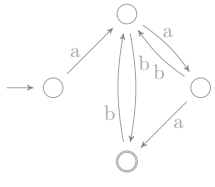
To decide the existence of of a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ we guess a factoring of the above form. First we show how to decide the existence of individual even- and odd-indexed runs, which we later combine to decide the existence of entire factorings.

We begin by showing how to calculate the odd-indexed runs. Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be configurations of C^γ between two successive parameters:

$$\gamma(p_i) < \text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}) < \gamma(p_{i+1}).$$

Consider a counter machine C_i, which is obtained from C by evaluating all comparisons as if the counter was between $\gamma(p_i)$ and $\gamma(p_{i+1})$. Formally, C_i is obtained from C by

- removing all transitions of the form $\geq p_j$ for $j > i$,
- removing all transitions of the form $\leq p_k$ for $k \leq i$,
- all transitions $\leq p_j$ for $j > i$ are replaced by $+0$ transitions in C_i,



- all transitions $\geq p_k$ for $k \leq i$ are replaced by $+0$ transitions in C_i,
- for $i > 0$ and $c \in \mathbb{N}$ we also remove all $\leq c$ transitions from C_i.

Note that the definition of C_i's depends only on the order of parameters in γ. Then provided the counter value stays between $\gamma(p_i)$ and $\gamma(p_{i+1})$ the runs in C_i^γ and C^γ coincide.

Recall that for a one-counter machine Z, configurations $\boldsymbol{c}, \boldsymbol{d}$ of Z and numbers $x, y \in \mathbb{N}$, we write $(\boldsymbol{c}, \boldsymbol{d}) \in Z(x, y)$ if there is a run $\pi : \boldsymbol{c} \to \boldsymbol{d}$ such that the counter stays between x and y, i.e., $x < \text{counter}(\pi) < y$. Then we have:

Lemma 6.1.2. *Let γ be an assignment with $\gamma(p_i) < \gamma(p_{i+1})$ for every i. Let $\boldsymbol{c}, \boldsymbol{d}$ be configurations with $\gamma(p_i) < \text{counter}(\boldsymbol{c}), \text{counter}(\boldsymbol{d}) < \gamma(p_{i+1})$. Then*

$$(\boldsymbol{c}, \boldsymbol{d}) \in C^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff (\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1}))$$

Proof. Immediate from the definition of C_i. □

Now, consider an arbitrary run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in C_i. During the run, the counter value can become less than $\gamma(p_i)$ or greater than $\gamma(p_{i+1})$. So π does not necessarily correspond to a run in C. However, notice that C_i is a one-counter machine without parameters or $\leq x$ constraints, i.e. an ordinary one-counter machine. Thus C_i has the following property [LLT04]: If there is a run between two configurations then there is a run between the same configurations such that the run does not deviate much from the initial and the final counter value. Formally:

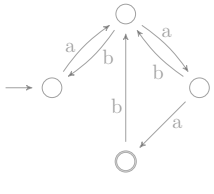
Lemma 6.1.3 ([LLT04], Lemma 42)**.** *Let X be a one-counter machine. There is a constant M (polynomial in $|X|$ and the magnitude of the largest constant appearing in X) such that for any configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ of X the following holds:*

- *Let $U = \min(\text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}))$ and $V = \max(\text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}))$.*
- *If $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ then there is a run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ such that $U - M \leq \text{counter}(\pi) \leq V + M$.*

So as long as $\gamma(p_1) + M < \text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}) < \gamma(p_2) - M$, the runs $\boldsymbol{c_1} \to \boldsymbol{c_2}$ in C_i correspond to runs in C.

Lemma 6.1.4. *Let γ be an assignment with $\gamma(p_i) + M < \gamma(p_{i+1})$ for all i. Let $\boldsymbol{c}, \boldsymbol{d}$ be configurations with $\gamma(p_i) + M < \text{counter}(\boldsymbol{c}), \text{counter}(\boldsymbol{d}) < \gamma(p_{i+1}) - M$. Then*

$$(\boldsymbol{c}, \boldsymbol{d}) \in C^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff \boldsymbol{c} \to^* \boldsymbol{d} \text{ in } C_i^\gamma.$$



Proof. Since C_i simulates C in the interval $(\gamma(p_i), \gamma(p_{i+1}))$ it is obvious that the biimplication holds if we restrict to runs inside the interval $(\gamma(p_i), \gamma(p_{i+1}))$ (Lemma 6.1.2):

$$(\boldsymbol{c}, \boldsymbol{d}) \in C^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff (\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1}))$$

It thus suffices to show that

$$(\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff \boldsymbol{c} \to^* \boldsymbol{d} \text{ in } C_i^\gamma$$

The left-to-right implication is immediate. For the converse, suppose that $\boldsymbol{c} \to^* \boldsymbol{d}$ in C_i^γ. Since, $\gamma(p_i) + M < \text{counter}(\boldsymbol{c}), \text{counter}(\boldsymbol{d}) < \gamma(p_{i+1}) - M$, by Lemma 6.1.3, there exists a run π in C_i^γ from $\boldsymbol{c}$ to $\boldsymbol{d}$ such that $\gamma(p_i) < \text{counter}(\pi) < \gamma(p_{i+1})$. Hence $(\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1}))$ as required. □

For the even-indexed runs, the reachability around individual parameters, i.e. in intervals $(\gamma(p_i) - M, \gamma(p_i) + M)$, can be precomputed. Suppose that $\gamma(p_{i-1}) < \gamma(p_i) - M < \gamma(p_i) + M < \gamma(p_{i+1})$ so that the interval $(\gamma(p_i) - M, \gamma(p_i) + M)$ does not contain $\gamma(p_{i-1})$ or $\gamma(p_{i+1})$. Let $-M < x, y < M$ and let π be a run from $(s, \gamma(p_i) + x)$ to $(t, \gamma(p_i) + y)$ such that $\gamma(p_i) - M \leq \text{counter}(\pi) \leq \gamma(p_i) + M$. Then for every component $\pi(j)$, we can write $\text{counter}(\pi(j)) = \gamma(p_i) + z_j$ for some $-M \leq z_j \leq M$. But now, the run π is valid for any specific value of $\gamma(p_i)$ as only z_j determines which transitions are enabled in C^γ.

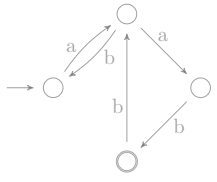
Lemma 6.1.5. *Let γ, δ be parameter assignments with $\gamma(p_i) + M < \gamma(p_{i+1}), \delta(p_i) + M < \delta(p_{i+1})$ for all i. Let $s, t \in c$ be states and $-M < x, y < M$ integers. Then*

$$\begin{aligned}((s, \gamma(p_i) + x), (t, \gamma(p_i) + y)) \in C^\gamma(\gamma(p_i) - M, \gamma(p_i) + M) \iff \\ ((s, \delta(p_i) + x), (t, \delta(p_i) + y)) \in C^\delta(\delta(p_i) - M, \delta(p_i) + M)\end{aligned}$$

Furthermore, it is decidable in polynomial time whether $((s, \gamma(p_i) + x), (t, \gamma(p_i) + y)) \in C^\gamma(\gamma(p_i) - M, \gamma(p_i) + M)$ for any (and all) such assignment γ.

Proof. For any run π from $(s, \gamma(p_i) + x)$ to $(t, \gamma(p_i) + y))$ in C^γ such that $\gamma(p_i) - M < \text{counter}(\pi) < \gamma(p_i) + M$ and for any index j we can write $\text{counter}(\pi(j)) = \gamma(p_i) + z_j$ for some $-M \leq z_j \leq M$. But now, the run π is valid for any specific value of $\gamma(p_i)$ as only z_j determines which transitions are enabled in C^γ.

Thus consider the graph G with vertices (s, z) where s is a state of C and $-M < z < M$. There is an edge from (s, z) to (s', z') in G if and only if there is an edge $(s, z' - z, s')$ in C, which goes from s to s' and updates the counter appropriately.



Similarly, if there is an edge $(s, \leq p_i, s')$ in C then we add an edge $(s, z) \to (s', z)$ for all $z \leq 0$. Similarly for other cases.

Now, any run in C completely contained in $(\gamma(p_i) - M, \gamma(p_i) + M)$ corresponds to a path in G and vice versa. Recall, Lemma 6.1.3, that M is polynomial in $|C|$. Thus, $|G|$ is also polynomial in $|C|$.

Therefore, for every pair of states s and t and values z, z' in $(-M, M)$, we can calculate using a standard graph algorithm (e.g. breadth-first search), whether there is a path from $(s, \gamma(p_i) + z)$ to $(t, \gamma(p_i) + z')$. □

Combining the previous lemmas on decidability of even- and odd-indexed runs, We now prove that the existence of a run between any two configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ is definable in the existential fragment of Presburger Arithmetic with Divisibility (∃PAD definable).

Theorem 6.1.6. *Given states $s, t \in C$ the set $G(C, s, t) = \{(x, y, n_1, \ldots, n_k) \mid (s, x) \to^* (t, y)$ in C^γ where $\gamma(p_i) = n_i\}$ is* ∃PAD *definable.*

Proof. First, consider the case such that $n_i + M < n_{i+1}$ for every i. The variable n_i denotes the value of $\gamma(p_i)$ in the constructed ∃PAD formulae. We encode the existence of a factoring $\pi = \pi_0 \to \pi_1 \to \pi_2 \to \cdots \to \pi_l$ as a ∃PAD formula.

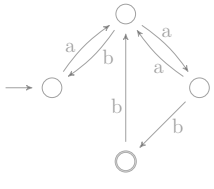
Note that $\text{start}(\pi_i)$ is always of the form $\text{start}(\pi_i) = (s_i, p_{f(i)} + x)$ for some $|x_i| \in \{M, M + 1\}$ and parameter $p_{f(i)}$. Hence, $\text{start}(\pi_i)$ is determined by the triple $(s_i, f(i), x_i)$. Similarly, $\text{last}(\pi_i)$ is determined by some triple $(t_i, g(i), y_i)$ with $|y_i| \in \{M, M + 1\}$.

Now, in Lemma 6.1.4 we showed that the odd-indexed runs π_{2i+1} correspond to runs in some one-counter machine $C_{h(2i+1)}$. By Lemma 2.7.1, the existence of a run in $C_{h(2i+1)}$ is ∃PAD expressible as:

$$\varphi_{2i+1} = \text{Reach}(C_{h(2i+1)}, s_{2i+1}, t_{2i+1})(n_{f(2i+1)} + x_{2i+1}, n_{g(2i+1)} + y_{2i+1}, n_1, \ldots, n_k)$$

where $h(2i+1)$ denotes the appropriate one-counter machine $C_{h(2i+1)}$ the run π_{2i+1} lies in.

In Lemma 6.1.5, we showed that the even-indexed runs are independent of γ, can be precomputed and the reachability relation can be hardwired into the formula. By taking a conjunction of the corresponding ∃PAD formulae we obtain a single ∃PAD formula. Precisely, given states $s_1, \ldots, s_l$ and $t_1, \ldots, t_l$ and offsets $x_1, \ldots x_l$ and $y_1, \ldots y_l$ and indices $f(1), \ldots, f(l)$ and $g(1), \ldots, g(l)$ and $h(1), \ldots h(l)$, consider the formula:



$$
\begin{aligned}
G(s,t,\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y},f,g,h)(x,y,\overrightarrow{n}) \;=\; & \bigwedge_i \varphi_{2i+1} \\
& \wedge\, s = s_1 \;\wedge\; n_{f(1)} + x_1 = x \\
& \wedge\, t = t_l \;\wedge\; n_{g(l)} + y_l = y \\
& \wedge\, \psi(f,g,h,\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y}) \\
& \bigwedge_i n_i + M < n_{i+1}
\end{aligned}
$$

The formula asserts that there is a run from (s,x) to (t,y) with the particular factoring. The conjunct $\psi(f,g,h,\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y})$ encodes that the even-indexed subruns are valid (precomputed using Lemma 6.1.5) and that odd- and even-indexed runs are adjacent (directly computable) and that the values of $f(i), g(i)$ and $h(i)$ are consistent (directly computable)[1]. The last conjunct encodes the restriction $\gamma(p_i)+M < \gamma(p_{i+1})$ from Lemmas 6.1.4 and 6.1.5.

This final restriction is relaxed as follows. If the parameters are not in the increasing order $\gamma(p_i) < \gamma(p_{i+1})$ then we build appropriate formulae by relabelling the parameters so that the resulting permutation of parameters is in increasing order.

If $\gamma(p_i) \leq \gamma(p_{i+1}) < \gamma(p_i)+M$ then note than M depends only on $|C|$ and so only finitely many possibilities exist for $\gamma(p_{i+1}) - \gamma(p_i)$. Hence we replace each occurrence of p_{i+1} in C by $p_i + w$ for the appropriate $w < M$.

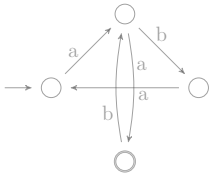
Now, there are only finitely many possibilities for $\overrightarrow{s}, \overrightarrow{t}, \overrightarrow{x}, \overrightarrow{y}, f, g$ and h. So all such formulae together define the reachability relation in C. Hence, the relation is $\exists$PAD definable. □

Recall that satisfiability of $\exists$PAD formulae is in NP [Lip81] and for a parametric timed automaton A with one parametric clock the corresponding one-counter machine C is exponential in size in $|A|$ (Lemmas 5.2.3 and 5.2.4). Hence, the halting problem for A is decidable in NEXP time (Theorem 5.2.5), which is optimal by Theorem 5.2.6.

6.2 Parametric Timed Automata with One Parameter

We now move to study parametric one-counter machines arising from parametric timed automata with two parametric clocks. The halting problem for the latter class

[1]Locally consistent means that $h(i)$ and $h(i+1)$ are consecutive, the indices $f(i)$ and $g(i)$ are consecutive and consistent with $h(i)$, etc.



is a long-standing open problem [AHV93]. We do not solve the problem in full. However, we use and develop some new counter-machines techniques to solve the problem in case of a single parameter.

So let A be a parametric timed automaton with two parametric clocks and a single parameter p. By Remark 5.2.9, if the corresponding parametric bounded one-counter machine C has an accepting run then it has one where the counter is bounded by $2 \cdot \gamma(p)$.

Now, notice that C has a single parameter but may contain '$\equiv 0 \bmod c_i$' or '$+[0,p]$' transitions. We first show how to decide the halting problem in case C has no '$\equiv 0 \bmod c_i$' or '$+[0,p]$' transitions. Later we show how to decide the halting problem if both types of transitions are allowed in C.

So let C be a simple parametric bounded one-counter machine (no '$\equiv 0 \bmod c$' or '$+[0,p]$' transitions) with a single parameter p. For given $k \in \mathbb{N}$ we decide the existence of an accepting run π such that $\text{counter}(\pi) \leq k \cdot \gamma(p)$ holds.[2] Now, for any such run π and index i we can write $\text{counter}(\pi(i)) = a\gamma(p) + b$ where $a \leq k$ and $b < \gamma(p)$. Since a is bounded, we build a one-counter machine G keeping a in the state space and b in the counter. We do not enforce $b < \gamma(p)$ (or any other $\leq x$ constraint) in G. Instead, we use Lemma 6.1.3 on G and split π into subruns close to and far from a multiple of $\gamma(p)$. We write $\pi = \tau_0 \to \pi_1 \to \tau_1 \ldots \pi_l \to \tau_l$ such that

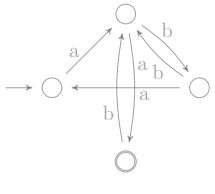
- for every τ_i the value $\text{counter}(\tau_i) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$,
- for every π_i we have $\text{counter}(\pi_i) \bmod \gamma(p) \in (M, \gamma(p) - M)$.

For a run ρ and a subset $S \subseteq \mathbb{N}$ of natural numbers, the notation $\text{counter}(\rho) \in S$ denotes the fact that for every i we have $\text{counter}(\rho(i)) \in S$.

Then we use techniques on factoring of runs analogous to those used for one parametric clock (Section 6.1) to build an ∃PAD formula encoding the existence of an accepting run. Decidability follows from the decidability of satisfiability of ∃PAD formulae. In general, we have:

Lemma 6.2.1. *Given C with one parameter p, no '$\equiv 0 \bmod c$' and no '$+[0,p]$' transitions, $k \in \mathbb{N}$ and states $s, t \in C$ the set $G(C, s, t, k) = \{(x, y, q) \mid \exists \pi : (s, x) \to (t, y) \in C^\gamma$ such that $\text{counter}(\pi) \leq k \cdot q$ where $q = \gamma(p)\}$ is ∃PAD definable.*

[2] $k = 2$ in case C was obtained by reduction from a parametric timed automaton with one parameter



Before proving the lemma, we first show how to decide the existence of τ_i's and π_i's. Given C and $k \in \mathbb{N}$ we introduce the one-counter machine C_k. Let S be the set of states of C. Then the states of C_k are $S \times \{0, \ldots, k-1\}$. Intuitively, if $z < \gamma(p)$, the configuration $((s,i),z)$ of C_k shall represents the configuration $(s, i \cdot \gamma(p) + z)$ of C. Machine C_k has the following transitions:

- $((s,i), \pm c, (t,i))$ if $(s, \pm c, t)$ is a transition in C,
- $((s,i), +0, (t, i \pm 1))$ if $(s, \pm p, t)$ is a transition in C,
- $((s,0), +0, (t,0))$ if $(s, \leq p, t)$ is a transition in C,
- $((s,i), +0, (t,i))$ for $i \geq 1$ if $(s, \geq p, t)$ is a transition in C.

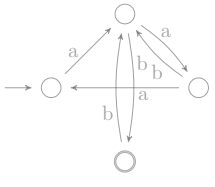
Intuitively, the "submachine" $S \times \{i\}$ should handle C when the counter of C lies in $[i\gamma(p), (i+1)\gamma(p)]$. Now, the simulation is not perfect, as the counter of G can be larger than $\gamma(p)$ thereby incorrectly enabling/disabling various transitions. However, note that G has no parameters or $\leq p$ transitions, i.e it is an ordinary one-counter machine. Hence, by Lemma 6.1.3, there is a constant $M \in \mathbb{N}$ such that we can assume that runs deviate by at most M from the initial and the final counter values.

Lemma 6.2.2. *Let γ be an assignments, s_1, s_2 states of C and $0 \leq a_1, a_2 < k$ and $M < b_1, b_2 < \gamma(p) - M$ be natural numbers. Then the following are equivalent.*

- *There is a run π from $(s_1, a_1\gamma(p) + b_1)$ to $(s_2, a_2\gamma(p) + b_2)$ in C^γ such that $\mathrm{counter}(\pi) \leq k\gamma(p)$ and $\mathrm{counter}(\pi) \bmod \gamma(p) \in (M, \gamma(p) - M)$.*
- *There is a run π' from $((s_1, a_1), b_1)$ to $((s_2, a_2), b_2)$ in C_k^γ.*

Proof. Analogous to lemma 6.1.4. Suppose that a run π exists. Then let f be a function that sends the configuration $(s, a\gamma(p) + b)$ of C^γ where $0 \leq b < \gamma(p)$ to the configuration $((s,a),b)$ of C_k^γ. By the construction of C_k and restrictions on π, the path $f(\pi)$ obtained by applying f to every component of π is a valid path in C_k^γ from $((s_1,a_1),b_1)$ to $((s_2,a_2),b_2)$.

Conversely, suppose that there is a run π'. By Lemma 6.1.3, we can assume that $0 < \mathrm{counter}(\pi') < \gamma(p)$. So let g be a function that sends the configuration $((s,a),b)$ of C_k^γ to the configuration $(s, a\gamma(p) + b)$ of C^γ. By the construction, the path $g(\pi')$ obtained by applying g to every component of π' is a valid path in C^γ from $(s_1, a_1\gamma(p) + b_1)$ to $(s_2, a_2\gamma(p) + b_2)$. □



The above lemma shows how to handle π_i runs. For the τ_i runs, recall that for every τ_i we have $\text{counter}(\tau_i) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. As in Lemma 6.1.5, the existence of runs τ_i is independent of $\gamma(p)$ and can be precomputed. Consider the graph with vertices $\{0, \ldots, k\} \times \{-M, \ldots, M\}$. Each vertex (i, j) corresponds to counter value $i \cdot \gamma(p) + j$ in C^γ. The edges mimic the transitions in C_k. Now, the graph is independent of $\gamma(p)$ and so we can calculate reachability between any pair of vertices.

Lemma 6.2.3. *Let γ be an assignment. Given $k \in \mathbb{N}$ and states s_1, s_2 of C and numbers $0 \leq a_1, a_2 \leq k$ and $b_1, b_2 \in [0, \ldots, M] \cup [\gamma(p) - M, \ldots, M)$ it is decidable in polynomial time whether there is a path π from $(s_1, a_1\gamma(p) + b_1)$ to $(s_2, a_2\gamma(p) + b_2)$ in C^γ such that* $\text{counter}(\pi) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. *Moreover, the existence of such a path is independent of $\gamma(p)$.*

We now have all the pieces necessary to prove Lemma 6.2.1.

Proof. (of Lemma 6.2.1)

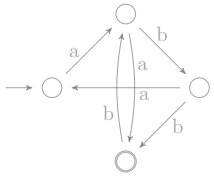
The proof is analogous to the proof of Theorem 6.1.6. Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be two configurations of C^γ. Consider a shortest run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in C^γ. Then π can be split into subruns close to and far from a multiple of $\gamma(p)$. We write $\pi = \tau_0 \to \pi_1 \to \tau_1 \ldots \pi_l \to \tau_l$ such that for every τ_i the value $\text{counter}(\tau_i) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. And for every π_i we have $\text{counter}(\pi_i) \bmod \gamma(p) \in (M, \gamma(p) - M)$.

Notice that the configuration $\text{start}(\tau_i)$ is uniquely determined by the state of C, $\lfloor \text{counter}(\text{start}(\tau_i))/\gamma(p) \rfloor$ and $\text{counter}(\text{start}(\tau_i)) \bmod \gamma(p)$. Now, C has only finitely many states, $\lfloor \text{counter}(\text{start}(\tau_i))/\gamma(p) \rfloor \leq k$ and $\text{counter}(\text{start}(\tau_i)) \bmod \gamma(p)$ can have only one of $2M+1$ values. Recall that π is a shortest run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$. In particular, π visits each configuration of C^γ at most once. Hence there are only $O(nkM)$ different initial configurations for τ_i's and hence $l = O(nkM)$.

Now, by Lemma 6.2.2, the existence of π_i can be witnessed by C_k. Further, by Lemma 6.2.3, the existence of runs τ_i is independent of $\gamma(p)$ and can be precomputed. The runs in C_k are $\exists$PAD expressible (Lemma 2.7.1). By taking a conjunction of the corresponding $\exists$PAD formulae we obtain a single $\exists$PAD formula

$$\left(\bigwedge_i \text{Reach}(C_k, \text{start}(\pi_i), \text{last}(\pi_i))(\text{counter}(\text{start}(\pi_i)), \text{counter}(\text{last}(\pi_i)), p) \right) \wedge \psi$$

defining the reachability relation for a particular factoring where (as in Theorem 6.1.6) the formula ψ encodes that τ_i's are valid (directly computable) and that τ_i and π_i can be connected by an edge (directly computable).



Since there are only finitely many initial and final states of π_i's and τ_i's, which uniquely determine the factoring, by taking the set of all such ∃PAD formulae, we conclude that the reachability relation is ∃PAD definable. □

6.2.1 Elimination of '$\equiv 0 \bmod c$' Transitions

Next, we show how to handle '$\equiv 0 \bmod c$' transitions. Let C be a parametric bounded one-counter machine with '$\equiv 0 \bmod c$' transitions. We now show how to eliminate '$\equiv 0 \bmod c$' transitions from C.

Let $K = \{c_1, \ldots, c_r\}$ be the set of all constants appearing as '$\equiv 0 \bmod c_i$' in C. Intuitively, we modify C to store in its state space the counter modulo each c_i. However, knowledge of $p \bmod c_i$ for each i is necessary for that.

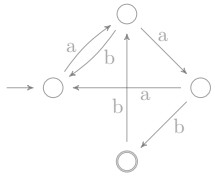
Given $D = (d_1, \ldots, d_r)$, let C_D be the parametric bounded one-counter machine which is obtained from C and which tracks the counter modulo each c_i assuming that $p \equiv d_i \bmod c_i$. Formally, the states of C_D are $S \times \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$ where S are the states of C and $\mathbb{Z}_{c_i}$ denotes the ring of integers modulo c_i. Let $(v_1, \ldots, v_r) \in \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$. Then C_D contains the following transitions:

- $((q, v_1, \ldots, v_r), \pm c, (q', v_1 \pm c, \ldots, v_r \pm c)$ if $(q, \pm c, q')$ is a transition in C,
- $((q, v_1, \ldots, v_r), \pm p, (q', v_1 \pm d_1, \ldots, v_r \pm d_r)$ if $(q, \pm p, q')$ is a transition in C,
- $((q, v_1, \ldots, v_r), +0, (q', v_1, \ldots, v_r))$ if $v_i = 0$ and $(q, \equiv 0 \bmod c_i, q')$ is a transition in C,
- $((q, v_1, \ldots, v_r), G, (q', v_1, \ldots, v_r))$ if (q, G, q') is a transition in C and G is a guard (comparison).

Notice that there are no '$\equiv 0 \bmod c$' transitions in C_D. By construction, runs in C_D^γ are equivalent to runs C^γ provided $d_i \equiv \gamma(p) \bmod c_i$. That is:

Lemma 6.2.4. *Let C be a parametric bounded one-counter machine with a single parameter p, let γ be an assignment such that $\gamma(p) = d_i \bmod c_i$ for each i. Let $(s, x), (t, y)$ be configurations of C. Then $(s, x) \to^* (t, y)$ in C^γ if and only if $((s, x \bmod c_1, \ldots, x \bmod c_r), x) \to^* ((t, y \bmod c_1, \ldots, y \bmod c_r), y)$ in C_D^γ.*

Proof. Let π be a run $\pi : (s, x) \to (t, y)$. Define $f : \mathbb{N} \to Z$ by $f(v) = (v \bmod c_1, \ldots, v \bmod c_r)$. Let τ be the run obtained from π by sending each configuration (s_i, v_i) to $(s_i \times f(v_i), v_i)$. By construction, each transitions $(s_i \times f(v_i), v_i) \to (s_{i+1} \times f(v_{i+1}), v_{i+1})$ is valid in C_D^γ.



For the converse, for configuration $\boldsymbol{c} = ((s_i, v_1, \ldots, v_r), w_i)$ let $g(c)$ be the function that gives $g(c) = (s_i, w_i)$. Let $\pi : ((s, x \mod c_1, \ldots, x \mod c_r), x) \to ((t, y \mod c_1, \ldots, y \mod c_r), y)$ be a run in C_D^γ. Let τ be the run obtained by applying g to π. By construction, each transitions $(s_i, w_i) \to (s_{i+1}, w_{i+1})$ is valid in C_D^γ. □

Using the above lemma together with the result (Theorem 6.2.1) for machines without '$\equiv 0 \mod c$' transitions we obtain:

Theorem 6.2.5. *Let C be parametric bounded one-counter machine with '$\equiv 0 \mod c_i$' transitions and a single parameter p. Given $k \in \mathbb{N}$ and states s and t, the set $H(C, s, t, k) = \{(x, y, q) \mid \exists \pi : (s, x) \to (t, y) \in C^\gamma$ such that $\text{counter}(\pi) \leq k \cdot q$ where $q = \gamma(p)\}$ is $\exists$PAD definable.*

Proof. For a fixed $c \in \mathbb{N}$ the $\exists$PAD formula

$$\varphi_c(x, y) = (\exists q \,.\, c \cdot q + y = x) \wedge y < c$$

asserts that $x \equiv y \mod c$.

Thus, given $D \in \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$, we construct the formula $H'(A, s, t, k, D)(x, y, p)$ asserting that D is consistent with p and that $(s, x) \to^* (t, y)$ in C_D:

$$\begin{aligned} H'(C, s, t, k, D)(x, y, p) = \exists \overrightarrow{v}(r), \overrightarrow{w}(r) \quad . \quad & G(C_D, s \times v, t \times w, k)(x, y, p) \wedge \\ & \bigwedge_i (x \equiv v(i) \mod c_i) \\ & \bigwedge_i (y \equiv w(i) \mod c_i) \\ & \bigwedge_i (p \equiv D(i) \mod c_i) \end{aligned}$$

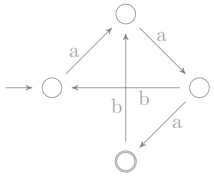
Since $\exists$PAD definable sets are closed under finite union and there are only finitely many possible D's, we get the desired result.

□

6.2.2 Elimination of '$+[0, p]$' Transitions

In this subsection suppose that the parametric bounded one-counter machine C contains '$\equiv 0 \mod c$' as well as '$+[0, p]$' transitions. We show that the reachability relation of C is $\exists$PAD definable also in this case.

Let $K = \{c_1, \ldots, c_k\}$ be the set of all constants appearing in '$\equiv 0 \mod c_i$' transitions in C and let $R = \text{lcm}(K)$ be their least common multiple. Suppose that C^γ



is nonempty and take π to be a shortest accepting run in C^γ. We will show that almost all traversals of '$+[0,p]$' edges increment the counter by at most R or by at least $\gamma(p) - R$.

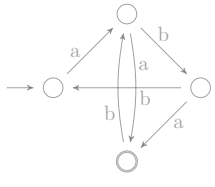
So suppose that at least two '$+[0,p]$' transitions appear along π. Then, we can write π as $\pi = \tau_1 \xrightarrow{e_1} \tau_2 \xrightarrow{e_2} \tau_3$ where e_1 and e_2 are '$+[0,p]$' transitions.

Denote the counter updates at e_1 and e_2 by f_1 and f_2, respectively. If $R < f_1, f_2 < \gamma(p) - R$ then consider the run τ_2' which is obtained from τ_2 by incrementing the counter by R. That is, for every index i, we have $\text{state}(\tau_2(i)) = \text{state}(\tau_2'(i))$ and $\text{counter}(\tau_2'(i)) = \text{counter}(\tau_2(i)) + R$. This corresponds to incrementing the counter in e_1, e_2 by $f_1 + R, f_2 - R$ respectively.

Now, τ_2' might not be a valid run. In particular, three things can potentially occur.

- There is a transition $(q, \text{`} \equiv 0 \mod c', q')$ from $\tau_2'(i)$ to $\tau_2'(i+1)$ for some i and $\text{counter}(\tau_2'(i)) \not\equiv 0 \mod c$. However, as $\tau_2(i) \equiv 0 \mod c$ by assumption on π and $c|R$, we have $c|\tau_2(i) + R = \tau_2'(i)$ and so this situation cannot happen.
- There is a transition $(q, \leq p, q')$ from $\tau_2'(i)$ to $\tau_2'(i+1)$ for some index i and $\text{counter}(\tau_2'(i)) > \gamma(p)$. Hence, $\text{counter}(\tau_2(i)) > \gamma(p) - R$. By assumption, we also have $\text{counter}(\tau_2(i)) \leq \gamma(p)$. Therefore, $\gamma(p) - R < \text{counter}(\tau_2(i)) \leq \gamma(p)$. But by minimality, π visits every configuration at most once and so such a situation can occur at most nR times—R times per each of n states. Otherwise, we would obtain a shorter run by cutting out the subrun between two occurrences of configuration $\tau_2(i)$.
- there exists index i such that $\text{counter}(\tau_2'(i)) > k \cdot \gamma(p)$. But then we have $k \cdot \gamma(p) - R < \text{counter}(\tau_2(i)) \leq k \cdot \gamma(p)$. By minimality, π visits every configuration at most once and so such a situation can occur at most nR times—R times per each of n states.

The above observations give rise to the following procedure. Let e_1 be the first '$+[0,p]$' transition that can be incremented by R such that there is a '$+[0,p]$' transition then can be decremented by R. Let π' be the run $\pi' = \tau_1 \xrightarrow{e_1} \tau_2' \xrightarrow{e_2} \tau_3$. Replace π by π' and repeat the process creating the runs $\pi', \pi'', \pi''', \ldots$. Note that $\text{counter}(\pi')$ is lexicographically larger than $\text{counter}(\pi)$. As π is of finite length and all counters can be incremented by at most $|\pi|\gamma(p)$, the procedure eventually terminates for some $\pi^{(k)}$. Let $\{f_1, \ldots, f_v\}$ be the set of increments caused by '$+[0,p]$' transitions along $\pi^{(k)}$. By above, each f_i is either at most R, at least $\gamma(p) - R$ or one of at most nR different values. Thus, we have shown:



Lemma 6.2.6. *Let C be a parametric bounded one-counter machine with a single parameter p and $\gamma : \{p\} \to \mathbb{N}$ be an assignment. If there is an accepting (bounded) run in C^γ then there is an accepting (bounded) run π such that at most nR counter increments by '$+[0,p]$' transitions that are not in the set $\{+0, +1, \ldots, +R, +p-R, \ldots, +p\}$. That is,*

$$|\{i \mid \pi(i) = +[0,p] \text{ and } R < \text{counter}(\pi(i+1)) - \text{counter}(\pi(i)) < \gamma(p) - R\}| \leq 2nR.$$

Proof. The case of runs bounded by a multiple of $\gamma(p)$ is covered in the paragraphs above.

If a run is not bounded by a multiple of $\gamma(p)$, then note that the third condition above does not occur. Hence, the cardinality of the set under consideration is at most nR. □

We now show that the existence of such a factoring can be specified by a ∃PAD formula.

Theorem 6.2.7. *Given C with '$\equiv 0 \bmod c$' and '$+[0,p]$' transitions and a single parameter p. Let $s, t \in C$ be states. Then the set $I(C,s,t,k) = \{(x,y,q) \in \mathbb{N}^3 \mid \exists \pi : (s,x) \to^* (t,y)$ in C^γ such that $\text{counter}(\pi) \leq k \cdot \gamma(p)$ where $\gamma(p) = q\}$ is ∃PAD definable.*

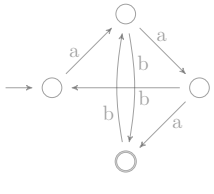
Proof. Let Z be the parametric bounded one-counter machine obtained from C by replacing each '$+[0,p]$' transition by $2R+2$ transitions: $+0, +1, \ldots, +R, +p-R, \ldots, +p$. Then $\pi^{(k)}$ (as defined in a paragraph above) can be factored as $\pi^{(k)} = \pi_0 \to \pi_1 \to \cdots \to \pi_v$ where each π_i is a run in Z^γ and there is a '$+[0,p]$' transition between π_i and π_{i+1}.

Given $v \leq 2nR$ and states $s = s_1, s_2, \ldots, s_v, t_1, \ldots, t_{v-1}, t_v = t$ the formula

$$I'(C,s,t,v,k,\overrightarrow{s},\overrightarrow{t})(x,y,p) = \exists \overrightarrow{x}, \overrightarrow{y}. \quad \bigwedge_{i=1\ldots v} I(Z,s_i,t_i,k)(x_i,y_i,p) \wedge x_1 = x \wedge y_v = y \quad \bigwedge_{i=1\ldots v-1} (0 \leq y_{i+1} - x_i \leq p \wedge E(t_i, \text{`}+[0,p]', s_{i+1})$$

asserts that such a factoring of length v exists where $\text{start}(\pi_i) = (s_i, x_i)$ and $\text{last}(\pi_i) = (t_i, y_i)$. The predicate $E(q, \text{`}+[0,p]', q')$ encodes that there is a '$+[0,p]$' transition between q and q' in C.

Since $v \leq 2nR$ there are only finitely many possibilities for $\overrightarrow{s}$ and $\overrightarrow{t}$. As ∃PAD sets are closed under finite union, the result follows. □



	simple	'$\equiv 0 \bmod c$'	'$\equiv 0 \bmod c$', '$+[0,p]$'
bounded runs	Theorem 6.2.1	Theorem 6.2.5	Theorem 6.2.7
unbounded runs	Theorem 6.3.14	Theorem 6.3.16	Theorem 6.3.17

Table 6.1: Parametric Bounded One-Counter Machines, One Parameter Case

As an immediate corollary (using Remark 5.2.9) we get the following result on parametric timed automata:[3]

Theorem 6.2.8. *The halting problem is decidable for parametric timed automata with two parametric clocks and a single parameter.*

6.3 Parametric Bounded One-Counter Machines with One Parameter

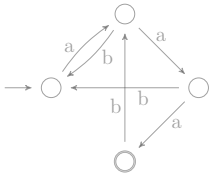
In the previous section we showed how to decide the halting problem for parametric timed automata with two parametric clocks with a single parameter. The decidability proof works by reduction to parametric bounded one-counter machines with a single parameter p and accepting runs with $\text{counter}(\pi) \leq 2\gamma(p)$. We now relax the restriction on $\text{counter}(\pi) \leq 2\gamma(p)$.

Let C be a parametric bounded one-counter machine with a single parameter. We show how to decide the halting problem for C. Note that by Theorem 5.2.7, machine C has the halting problem equivalent to some parametric timed automaton with two parametric clocks with *two* parameters. However, the decidability of the halting problem for the latter class is open in general. Thus, solving decidability in this special class of one-counter machines is a necessary step towards resolving the general case of the halting problem for parametric timed automata. See the Discussion section for more detail.

We begin by considering the situation when C has neither '$\equiv 0 \bmod c$' nor '$+[0,p]$' transitions. The results for the various cases of the single parameter problem are summarised in Table 6.1.

The decidability of the halting problem for C is quite involved and depends on the theory of ordinary parametric one-counter machines [Haa12] (summarised in Section 6.3.3). We begin by giving an overview of the proof.

[3] In fact, it can be shown in case of a single parameter that all '$+[0,p]$' transitions in C can be trivially eliminated and thus the result follows already from Theorem 6.2.5.



6.3.1 Proof Outline For Simple Parametric Bounded One-Counter Machines

We first show the result for simple parametric bounded one-counter machines (no '$\equiv 0 \bmod c$' or '$+[0,p]$' transitions). To show that such a machine halts C, we have to find an assignment γ and an accepting run π in C^γ. So suppose that there is some γ, consider any accepting run π in C^γ and factor π as follows

$$\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t \tag{6.1}$$

where $\text{counter}(\tau_i) \leq 2\gamma(p) < \text{counter}(\pi_i)$ for every i. Notice that since $\text{counter}(\pi_i) \geq 2\gamma(p)$, the runs π_i's do not use any $\leq p$ transitions. Intuitively, such runs correspond to runs in some one-counter machine. Using theory of one-counter machines developed in [Haa12, HKOW09], each individual π_i can be factored in a special way. Building upon that theory, we show in Section 6.3.2 that there is $K \in \mathbb{N}$ depending only on C (and not on γ) such that all but a bounded number of π_i's have $\text{counter}(\pi) \leq K \cdot \gamma(p)$.

Rewriting (6.1), we then have that there are $K, L \in \mathbb{N}$ depending only on C (and not on γ) such that $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ and $t \leq L$ and $\text{counter}(\tau_i) \leq K \cdot \gamma(p) < \text{counter}(\pi_i)$ for every i.

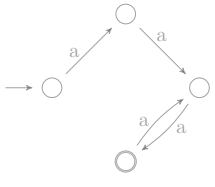
In Theorem 6.3.14 we use results from [Haa12, HKOW09] and Theorem 6.2.1 to show that both π_i's and τ_i's are $\exists$PAD definable, respectively. Hence, $\exists$PAD definability of reachability in C follows.

For the rest of the section, fix a simple parametric bounded one-counter machine C with a single parameter p. Denote the number of states of C by n. We now show how to obtain a factoring of runs in C^γ.

6.3.2 Properties of Parametric Bounded One-Counter Machines

Let γ be an assignment and π a run in C^γ. As in (6.1), write π as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ such that $\text{counter}(\tau_i) \leq 2\gamma(p) < \text{counter}(\pi_i)$ for every i. The goal of this section is to show that irrespective of γ, all but finitely many of π_i's are bounded by $K \cdot \gamma(p)$ for some constant $K \in \mathbb{N}$ independent of γ and depending only on C.

Notice that since $\text{counter}(\pi_i) \geq 2\gamma(p)$, the runs π_i's do not use any $\leq p$ transitions. Hence, we can think of them as runs in a one-counter machine. The machine $C_{>p}$ is obtained from C by removing all $\leq p, \leq c$ transitions and leaving all other transitions and states unchanged. Then $C_{>p}$ is a parametric one-counter machine without upper



bounds. Moreover, $C_{>p}$ simulates runs of C when the counter stays above $\gamma(p)$. Formally,

Lemma 6.3.1. *Let $s, t \in C$ be states of C and $0 < x, y \in \mathbb{N}$ be positive natural numbers. Then there is a run $\pi : (s, \gamma(p) + x) \to (t, \gamma(p) + y)$ in C^γ such that* $\text{counter}(\pi) > \gamma(p)$ *if and only if there is a run $\pi' : (s, \gamma(p) + x) \to (t, \gamma(p) + y)$ in $C^\gamma_{>p}$.*

Proof. Consider $\pi : (s, \gamma(p) + x) \to (t, \gamma(p) + y)$ in C^γ such that $\text{counter}(\pi) > \gamma(p)$. Then π does not use any $\leq p$ transitions. Hence, π is a run in $C^\gamma_{>p}$. The converse is symmetric. □

We now turn to study runs in parametric one-counter machines. In the following section we study properties of runs in one-counter machines that will be useful to bound subruns π_i's.

6.3.3 Properties of Runs in One-Counter Machines

Let Z be a one-counter machine (with no upper bounds and no parameters). The reachability problem for such machines was shown decidable in [HKOW09, Haa12] and a result of [HKOW09, Haa12] also characterised the reachability between any two configurations. We now outline the results obtained in those publications.

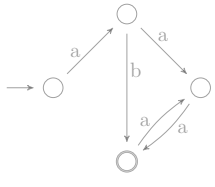
Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be two configurations of Z. We aim to show that if there is a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ then there is one of a special form. So suppose that there is a run π from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$.

Further, suppose that there is loop with positive net effect followed by a loop with negative net effect in π. Let α be the first loop in π with a positive net effect and let β be the last loop in π with negative net effect: $\text{effect}(\alpha) > 0 > \text{effect}(\beta)$. Then π can be written as:

$$\pi = \pi_1 \to \alpha \to \tau \to \beta \to \pi_3.$$

Notice that for any $k \in \mathbb{N}$, we have $k|\,\text{effect}(\beta)|\,\text{effect}(\alpha) + k\,\text{effect}(\alpha)\,\text{effect}(\beta) = 0$. Therefore, the run $\pi_k = \pi_1 \to \alpha^{(1+k|\,\text{effect}(\beta)|)} \to \tau \to \beta^{(1+k|\,\text{effect}(\alpha)|)} \to \pi_3$ obtained from π by taking the the loop α exactly $1 + k|\,\text{effect}(\beta)|$ times and by taking the loop β exactly $1 + k\,\text{effect}(\alpha)$ times starts and finishes in the same configuration as π does. Moreover, as α is a positive loop, the counter along the run π_k never becomes negative and so π_k is a valid path in Z for every $k > 0$.

Therefore, the counter value in τ can be made arbitrarily large by pumping the counter arbitrarily high using α and then bringing the counter back using β. In



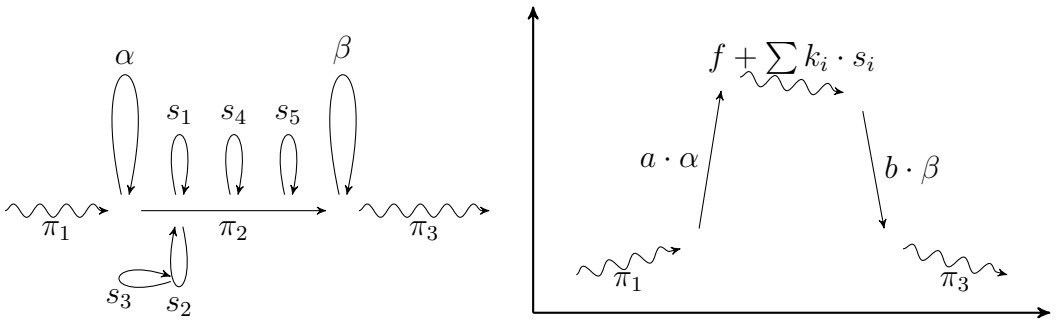


Figure 6.3: Factoring of a run as $\pi_1 \cdot \pi_2 \cdot \pi_3$ is shown on the left. The right figure shows how the counter value changes during the run. The loops α and β are positive and negative respectively. Thus, they can be used to pump up the counter arbitrarily high and then bring it back. Hence, the order of traversals of loops $s_1, \ldots, s_5$ does not matter.

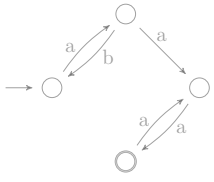
particular, we can ignore in τ the order in which individual loops are visited as we can assume that the counter is large enough so that no transition makes counter negative. Thus, the precise order of transitions in π is irrelevant as long as it gives one connected path. Such path are described as follows. (See also Figure 6.3)

Definition 6.3.2. *For simple loops s_i, path f and numbers k_i we say that path π is of the form $f + \sum_i k_i s_i$ if the expression describes how many times each transition of Z is visited by π. Formally, for each transition e in Z, we have* $\mathrm{count}(\pi, e) = \mathrm{count}(f, e) + \sum_i k_i \,\mathrm{count}(s_i, e)$ *where the expression* $\mathrm{count}(X, e)$ *denotes the number of occurrences of a transition e in the set X.*

Further, by removing nested negative loops from α, we can assume that α is a simple loop. Similarly, by removing nested positive loops from β, we can assume that β is a simple loop. Formally, the following result appeared in [HKOW09, Haa12].

Lemma 6.3.3 ([HKOW09, Haa12])**.** *Let Z be a one-counter machine (without upper bounds or parameters). For configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ of Z, if there is a run $\tau : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in Z then there is a run π from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ such that $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ and*

- $\min \mathrm{counter}(\pi) \geq \min \mathrm{counter}(\tau)$,
- *There are no positive simple loops in π_1,*
- *There are no negative simple loops in π_3,*
- $\pi_2 = \alpha^a \to \tau \to \beta^b$ *where α and β are simple loops such that* $\mathrm{effect}(\beta) < 0 < \mathrm{effect}(\alpha)$ *and τ is of the form $(f + \sum k_i s_i)$ where f is a simple path, s_i's are simple loops, $a, b, k_i \in \mathbb{N}$ and the set of edges $f \cup \{s_i\}_i$ is connected.*



We denote by type(π) *the tuple* (support(π_1), $\alpha, \beta, f, \{s_i\}$, support($\pi_3$)). *Runs of this form are called* factored. *If* $\pi_2 \neq \epsilon$ *then we say that* π *is* pumpable.

Note that there are only finitely many types. Moreover, $\operatorname{div}(\tau) \leq \operatorname{div}(f) + \sum_i k_i \operatorname{div}(s_i) \leq \operatorname{div}(f) + n \sum_i k_i M$ where M is the largest constant appearing in Z.

Now, in the middle segment π_2 only the number of traversal of loops matters as, by pumping α and β if necessary, we can always assume that the counter value is large enough. Therefore, the above Lemma has a converse stating that once the numbers of traversals are specified a run with the corresponding effect always exists[4]

Lemma 6.3.4 ([Haa12])**.** *Let* f *be a simple path,* s_i *be simple loops such that* $f \cup \{s_i\}_i$ *is connected. Let* π_1, π_3 *be runs such that* $\operatorname{last}(\pi_1) = \operatorname{first}(f)$ *and* $\operatorname{last}(f) = \operatorname{first}(\pi_3)$ *and let* $a, b, k_i > 0$ *be natural numbers. Then there is a run* π *in* Z *of the form* $\pi_1 \to \alpha^{a'} \to f + \sum_i k_i s_i \to \beta^{b'} \to \pi_3$ *such that* $\operatorname{effect}(\pi) = \operatorname{effect}(\pi_1) + a \operatorname{effect}(\alpha) + \operatorname{effect}(f) + \sum_i k_i \operatorname{effect}(s_i) + b \operatorname{effect}(\beta) + \operatorname{effect}(\pi_3)$.

In particular, by modifying number of traversals of loops s_i in a factored run it is always possible to create another factored run with the corresponding change in effect. Formally,

Lemma 6.3.5. *Let* $\pi = \pi_1 \to \alpha^a \to (f + \sum k_i s_i) \to \beta^b \to \pi_3$ *be a factored run. Let* $u, v \in \{a, b, k_1, k_2, \ldots\}$ *and* $\Delta_u, \Delta_v \in \mathbb{N}$ *then if* $u + \Delta_u, v + \Delta_v > 0$ *then there is a run* π'*, denoted as* $\pi' = \pi(u \leftarrow u + \Delta_u, v \leftarrow v + \Delta_v)$*, such that*[5]

- $\operatorname{effect}(\pi') = \operatorname{effect}(\pi) + \Delta_u \operatorname{effect}(\sigma_u) + \Delta_v \operatorname{effect}(\sigma_v)$,
- $\operatorname{counter}(\pi') \geq \min(\operatorname{counter}(\pi), \operatorname{counter}(\pi) + \Delta_u \operatorname{effect}(\sigma_u) + \Delta_v \operatorname{effect}(\sigma_v))$

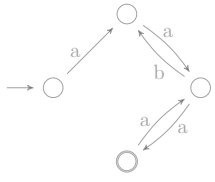
where σ_u, σ_v *are the loops associated with* u *and* v*, respectively.*

Proof. Let π' be the path as in the statement. Such a path exists by Lemma 6.3.4. Write π' as $\pi' = \rho_1 \rho_2 \rho_3$ where

- ρ_1 corresponds to the prefix $\pi_1 \to \alpha^{a'}$,
- ρ_2 corresponds to the middle segment $f + \sum k_i' \cdot s_i'$,
- ρ_3 corresponds to the suffix $\beta^{b'} \to \pi_3$.

[4] In detail, the run is a collection of Eulerian traversals of the graph spanned by f and $\{s_i\}_i$ with the appropriate multiplicities.

[5] The statement naturally generalise to modifying the number of traversals of any number of loops. However, in all the results in this thesis we modify exactly two loops and therefore present the result only in this special case.



Since up to possibly different number of traversals of the positive loop α the path ρ_1 agrees with a prefix of π, we have $\text{counter}(\rho_1) \geq \min \text{counter}(\pi)$.

Further, by modifying a' and b' if necessary, we can assume that $\min(\text{counter}(\rho_2))$ is sufficiently large and hence greater than $\min \text{counter}(\pi)$.

Finally, note that ρ_3 corresponds to a suffix of π where the counter is incremented by $\Delta_u \, \text{effect}(\sigma_u) + \Delta_v \, \text{effect}(\sigma_v)$. Hence, $\text{counter}(\rho_3) \geq (\min \text{counter}(\pi)) + \Delta_u \, \text{effect}(\sigma_u) + \Delta_v \, \text{effect}(\sigma_v)$.

By taking the minimum over the three cases above, the result follows. □

Going back to the factoring (6.1) of an accepting run π in the parametric bounded one-counter machine C^γ, we deduce from Lemmas 6.3.3 and 6.3.1 that every π_i can be factored.

Lemma 6.3.6. *Let c_1, c_2 be two configurations in C^γ. There is a run $\pi : c_1 \to c_2$ such that $\text{counter}(\pi) > 2\gamma(p)$ if and only if there is a factored run $\pi' : c_1 \to c_2$ such that $\text{counter}(\pi') > 2\gamma(p)$.*

Proof. The right-to-left implication is immediate. For the left-to-right implication, consider the one-counter machine $C_{>p}$. Now, π is a run in C^γ such that $\text{counter}(\gamma) > \gamma(p)$. Hence, π does not traverse any $\leq p$ transitions in C. Hence π is a run in $C^\gamma_{>p}$. Applying Lemma 6.3.3 to π and $C_{>p}$ gives the desired result. □

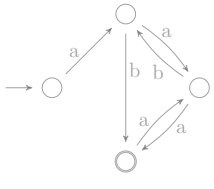
So we assume that every π_i in the factoring (6.1) is factored. In the following lemmas we study factored runs of different types in order to bound the factoring (6.1).

6.3.4 Properties of Factored Runs

In this section we prove tighter bounds on the structure of factored runs. Recall that in Lemma 6.3.3 we have associated a type with every run. In the following sections, we partition all factored runs according to their types into three classes (one class for nonpumpable runs and two classes for pumpable runs). We first show that all nonpumpable as well as one class of pumpable runs is essentially bounded. Then we show that the remaining pumpable runs can appear only a bounded number of times.

6.3.5 Nonpumpable Runs

First, consider nonpumpable runs in C^γ. The following lemma says that nonpumpable runs are bounded.



Lemma 6.3.7. *Let $\pi = \pi_1\pi_2\pi_3$ be a nonpumpable run in C^γ. Then* $\text{counter}(\pi) \leq n\gamma(p) + \max(\text{counter}(\text{first}(\pi)), \text{counter}(\text{last}(\pi)))$.

Proof. A simple path in C is always of length at most n and a transition increases the counter by at most $\gamma(p)$. Thus, along any simple path, the counter increases by at most $n\gamma(p)$. Now, π_1 has no positive loops. So $\text{div}(\pi_1) \leq n\gamma(p)$ and hence, $\text{counter}(\pi_1) \leq \text{counter}(\text{first}(\pi_1)) + n\gamma(p)$.

Similarly, starting from $\text{last}(\pi)$ and considering the reverse of the path π_3 we obtain $\text{counter}(\pi_3) \leq \text{counter}(\text{last}(\pi_3)) + n\gamma(p)$. Combining the two gives the result. □

6.3.6 Pumpable Runs

Next, consider pumpable runs. For such runs, the cross product between positive and negative loops occurring in π_2 will play a crucial role.

For a run π in C^γ, by counting the number of $\pm p$ and the net effect of $\pm c$ transitions in π there is a natural way of writing $\mathbf{effect}\,(\boldsymbol{\pi}) = \boldsymbol{a\gamma(p) + b}$ for $a, b \in \mathbb{Z}$ (see e.g., Figure 6.4 and Figure 6.5).

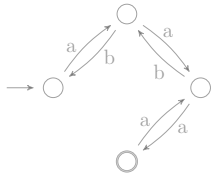
Let σ, τ be two runs and write $\text{effect}(\sigma) = a\gamma(p) + b$ and $\text{effect}(\tau) = c\gamma(p) + d$. By the *cross product* of σ, τ we mean $\sigma \times \tau = ad - bc$.

Definition 6.3.8. *Let T be a type of a factored run and let P and N be the set of positive and negative loops in T, respectively. We say that T (or a run π of type T) is* linear *if for every $\rho \in P$ and $\sigma \in N$ the cross product $\rho \times \sigma = 0$ equals zero. Otherwise, we say that T (or π) is* nonlinear.

Notice that for $\gamma(p)$ big enough (bigger than $2n$), the sign of the effect of a simple loop does not depend on $\gamma(p)$. Hence the distinction is well defined and does not depend on γ for all but finitely many cases. By modifying the number of traversals of individual loops we will show that linear runs are bounded (Lemma 6.3.9) and that nonlinear runs occur only bounded number of times (Lemma 6.3.12).

6.3.7 Linear Runs

We now show that linear runs are bounded by a multiple of $\gamma(p)$. For loops ρ and σ with $\text{effect}(\rho) = a \cdot \gamma(p) + b$ and $\text{effect}(\sigma) = c \cdot \gamma(p) + d$ the constraint $\rho \times \sigma = 0$ implies that the vectors $(a, b) \in \mathbb{N}^2$ and $(c, d) \in \mathbb{N}^2$ are collinear. Hence, there is $q = (q_1, q_2) \in \mathbb{Q}^2$ such that $(a, b) = k_1 \cdot q$ and $(c, d) = k_2 \cdot q$ for some $k_1, k_2 \in \mathbb{N}$. Hence, all loops can be expressed as $k \cdot q$ (see Figure 6.4). By using this observation



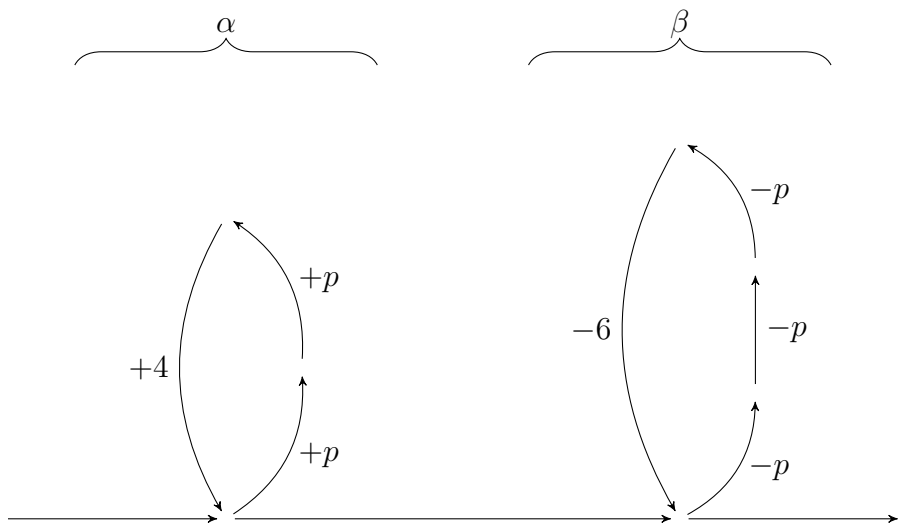


Figure 6.4: Automaton with only linear runs. The automaton has two loops. A positive loop α with effect$(\alpha) = 2p+4$ and a negative loop β with effect$(\beta) = -3p-6$. Notice that $\alpha \times \beta = 0$ and the common denominator of the loops is $p + 2$.

and modifying the number of traversals of loops s_i, we show that linear runs are bounded.

Lemma 6.3.9. *Assume $\gamma(p) > 2n$. Let $\pi : (s, x) \to (t, y)$ be a linear run in C^γ such that* counter$(\pi) > 2\gamma(p)$. *Then there is an equivalent run $\pi' \sim \pi$ such that*

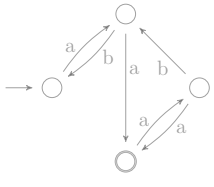
$$2\gamma(p) < \text{counter}(\pi') \leq \max(x, y) + V\gamma(p))$$

for some constant $V \in \mathbb{N}$ depending only on C (and not on γ).

Proof. Let π be factored as $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ where $\pi_2 = \alpha^a \to (f + \sum s_i p_i + \sum t_i n_i) \to \beta^b$ where p_i's are positive loops and n_i's are negative loops. Then Lemma 6.3.7 implies that counter(π_1), counter$(\pi_3) \leq \max(x, y) + n\gamma(p)$.

Let S be the set of loops in π_2 and take a loop $\sigma \in S$. Then we can uniquely write effect$(\sigma) = g\gamma(p) + h$ for some $g, h \in \mathbb{N}$. Consider the mapping $\rho : S \to \mathbb{Z}^2$ which sends σ to the point (g, h), i.e., $\rho(\sigma) = (g, h)$. Since $\sigma \times \tau = 0$ for all loops $\sigma, \tau \in S$, the mapping ρ sends all loops to a single line. In particular, there is a rational[6] point $q = (q_1, q_2) \in \mathbb{Q}^2$ such that for every $\sigma \in S$ there is an integer $k \in \mathbb{Z}$ such that $\rho(\sigma) = kq$. Thus, effect$(\sigma) = k(q_1\gamma(p) + q_2)$. We think of all the loops of S as being a multiple of $q_1\gamma(p) + q_2$. By a slight abuse of notation, we use $q = q_1\gamma(p) + q_2$ and write $\sigma = k \cdot q$. Note that q depends only on C (and not on γ).

[6] In fact, it can be shown that $q \in \mathbb{N}^2$



We can ignore the order in which transitions are traversed in π_2. Thus, we modify the number of traversals of individual loops in π_2 while keeping the net effect of the run unchanged. Write $\beta = xq$ and $n_i = y_i q$ for some $x, y_i \in \mathbb{N}$. If $t_i \geq x$ then

$$\begin{aligned}(t_i - x)\,\mathrm{effect}(n_i) + (y_i + b)\,\mathrm{effect}(\beta) &= (t_i - x)\,\mathrm{effect}(y_i q) + (y_i + b)\,\mathrm{effect}(xq) \\ &= t_i y_i\,\mathrm{effect}(q) - x y_i\,\mathrm{effect}(q) \\ &\quad + y_i x\,\mathrm{effect}(q) + bx\,\mathrm{effect}(q) \\ &= t_i\,\mathrm{effect}(y_i q) + b\,\mathrm{effect}(xq) \\ &= \mathrm{effect}(t_i n_i) + \mathrm{effect}(b\beta)\end{aligned}$$

So if $t_i > x$ then, by Lemma 6.3.5, there is a run π' of the form $\pi' = \pi(t_i \leftarrow t_i - x, b \leftarrow b + y_i)$ such that $\mathrm{counter}(\pi') > 2\gamma(p)$ and $\mathrm{effect}(\pi') = \mathrm{effect}(\pi)$. Repeating the construction if necessary, we can assume that $t_i \leq x$ for all $n_i \in N$. Thus, we just bounded the number of traversals of negative loops by $t_i \leq x$.

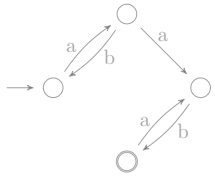
Similarly, write $\alpha = yq$ and $p_i = z_i q$ for some $a, z_i \in \mathbb{N}$. Then if $s_i > y$ then, by Lemma 6.3.5, there is a sun π' of the form $\pi(a \leftarrow a + z_i, s_i \leftarrow s_i - y)$ such that $\mathrm{effect}(\pi'') = \mathrm{effect}(\pi)$ and $\mathrm{counter}(\pi'') > 2\gamma(p)$. Repeating the construction if necessary, we can assume that $s_i \leq y$ for all $p_i \in P$. Thus, we just bounded the number of traversals of positive loops by $s_i \leq y$.

Denote $f + \sum s_i p_i + \sum t_i n_i$ by τ. Then we have

$$\begin{aligned}|\,\mathrm{div}(\tau)| &= \mathrm{div}(f + \sum s_i p_i + \sum t_i n_i) \\ &\leq n\gamma(p) + |S|y \cdot \mathrm{div}(q) + |S|x \cdot \mathrm{div}(q) \\ &\leq K\gamma(p)\end{aligned}$$

for some constant K depending on S, x and y. Notice that $\mathrm{div}(q) \leq n\gamma(p)$. Further, $|S|$ as well as x and y depend only on C (and not on γ).

Now, if $(a - x)\,\mathrm{effect}(\alpha) \geq K\gamma(p) \geq \mathrm{div}(\tau)$, then let $\pi''' = \pi_1 \to \alpha^{a-x} \to f + \sum_i k_i s_i \to \beta^{b-y} \to \pi_3$. By the assumption, π''' is a valid path and $\mathrm{effect}(\pi''') = \mathrm{effect}(\pi)$ and $\mathrm{counter}(\pi''') > 2\gamma(p)$. Hence, by repeated application if necessary, we can assume that $\mathrm{div}(\alpha^a) \leq K\gamma(p) + x\,\mathrm{div}(\alpha) \leq K\gamma(p) + xn\gamma(p)$ as α is a simple loop. Therefore, $\mathrm{div}(\alpha^a) \leq K'\gamma(p)$ for some constant $K' \in \mathbb{N}$ depending only on C. Hence,



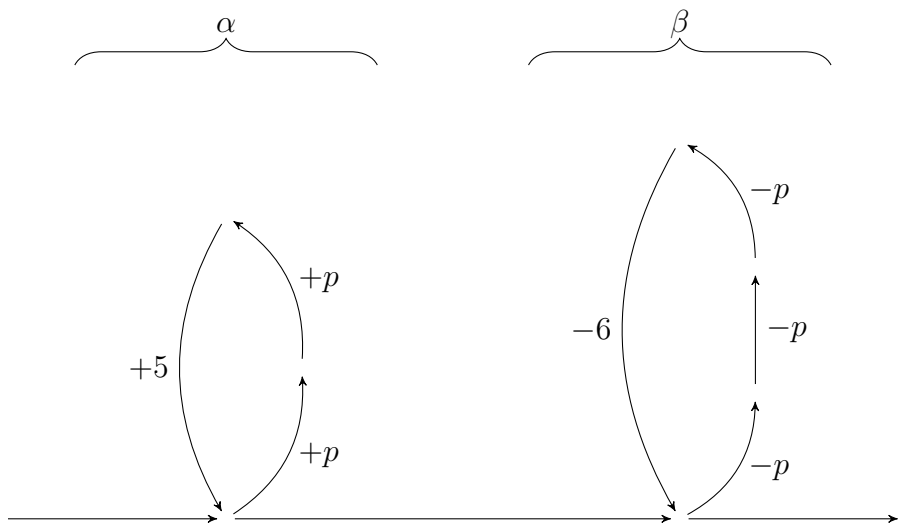


Figure 6.5: Automaton with non linear runs. The automaton has two loops. A positive loop α with effect$(\alpha) = 2p+5$ and a negative loop β with effect$(\beta) = -3p-6$. Notice that $\alpha \times \beta = -2 \cdot 5 - (-3 \cdot 5) = 3$ and that effect$(3\alpha) +$ effect$(2\beta) = 3 = \alpha \times \beta$.

we calculate

$$\begin{aligned}
\text{counter}(\pi) &\leq \max \text{counter}(\pi_1\pi_3) + \text{div}(\pi_2) \\
&\leq \max(x, y) + n\gamma(p) && \{\text{Lemma 6.3.7}\} \\
&\quad + \ \text{div}(\alpha^a) + |\,\text{div}(\tau)| + \text{div}(\beta) \\
&\leq \max(x, y) + n\gamma(p) + K'\gamma(p) + K\gamma(p) + n\gamma(p) \\
&\leq \max(x, y) + K''\gamma(p)
\end{aligned}$$

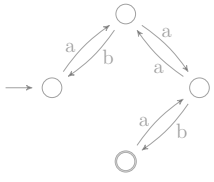
for some constant $K'' \in \mathbb{N}$ depending only on C (and not on γ).

□

6.3.8 Nonlinear Runs

The last class of runs are the nonlinear runs. In general, we cannot bound nonlinear runs. However, we show that we can always assume that any run in C^γ contains only finitely many nonlinear subruns.

Consider a nonlinear run from configuration (s, x) to configuration (t, y). By definition, there is a positive loop ρ and a negative loop σ in the run such that $\rho \times \sigma \neq 0$. By modifying the number of times ρ and σ are traversed, we obtain runs from (s, x) to $(t, y \pm L)$ for some $L \in \mathbb{N}$ depending only on $\rho \times \sigma$ (see Figure 6.5).



Repeating the process, we build nonlinear runs starting in (s, x) and finishing in configurations $(t, y + kL)$ for almost all valid $k \in \mathbb{Z}$.

Lemma 6.3.10. *Assume $\gamma(p) > 2n$. Let T be a nonlinear type. Then there exists a natural number $L(T) \leq n^4$ depending only on T such that the following property holds. For states $s, t \in C$ and counter values $x, y \in \mathbb{N}$,*

- *if there is a run $\pi : (s, x) \to (t, y)$ of type T in C^γ such that $\mathrm{counter}(\pi) > 2\gamma(p)$ and $x, y \leq 3\gamma(p)$*
- *then for every $k \in \mathbb{Z}$ with $y + kL(T) > 2\gamma(p)$ there is a run $\pi' : (s, x) \to (t, y + kL(T))$ in C^γ with $\mathrm{counter}(\pi') > \gamma(p)$.*

Proof. Write $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3 = \pi_1 \to \alpha^a \to f + \sum k_i s_i \to \beta^b \to \pi_3$. By assumption, π_2 contains a positive loop ρ and a negative loop σ such that $\rho \times \sigma \neq 0$. Write $\mathrm{effect}(\rho) = a\gamma(p) + b$ and $\mathrm{effect}(\sigma) = c\gamma(p) + d$. Then $c \leq 0 \leq a$ and

$$a \cdot \mathrm{effect}(\sigma) - c \cdot \mathrm{effect}(\rho) = a(c\gamma(p) + d) - c(a\gamma(p) + b) = ad - cb = \rho \times \sigma.$$

Hence, $a \cdot k \cdot \mathrm{effect}(\sigma) - c \cdot k \cdot \mathrm{effect}(\rho) = k \cdot \rho \times \sigma$ for any $k \in \mathbb{N}$. Denote $\rho \times \sigma$ by P and suppose that path π takes the loops ρ and σ exactly r and s times, respectively.

First, assume that $P > 0$. Then for any $k \in \mathbb{N}$, by Lemma 6.3.5, there is a path π_k of the form $\pi_k = \pi(s \leftarrow s + ak, r \leftarrow r + |c|k)$ such that $\mathrm{counter}(\pi_k) > 2\gamma(p)$ and $\mathrm{effect}(\pi_k) = \mathrm{effect}(\pi) + kP$. Thus, $\mathrm{counter}(\mathrm{last}(\pi_k)) = y + kP$ and so by changing the number of traversals of ρ and σ we can increase the counter by an arbitrary multiple of P.

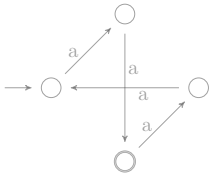
Recall that $\mathrm{effect}(\sigma) < 0$. Therefore, there exists the largest natural number $q \in \mathbb{N}$ such that $qP + \mathrm{effect}(\sigma) < 0$. Hence, $-P \leq qP + \mathrm{effect}(\sigma) < 0$. If we denote $qP + \mathrm{effect}(\sigma)$ by R then we have:

$$q\big[a \cdot \mathrm{effect}(\sigma) - c \cdot \mathrm{effect}(\rho)\big] + \mathrm{effect}(\sigma) = q(\rho \times \sigma) + \mathrm{effect}(\sigma) = qP + \mathrm{effect}(\sigma) = R.$$

For $k \in \mathbb{N}$ suppose that $y + kR > 2\gamma(p)$. Then as $3\gamma(p) > y$ it holds that $kR > -\gamma(p)$. Therefore, by Lemma 6.3.5, there is a path $\tau_k = \pi(s \leftarrow s + k(qa+1), r \leftarrow r + |c|qk)$ such that $\mathrm{effect}(\tau_k) = \mathrm{effect}(\pi) + kR$ and $\mathrm{counter}(\tau_k) + kR > 2\gamma(p) + kR > \gamma(p)$. Hence, $\mathrm{counter}(\mathrm{last}(\tau_k)) = y + kR$.

So let $L = \mathrm{lcm}(P, |R|)$ be the least common multiple of P and $|R|$. Then $L \leq PR \leq P^2 \leq n^4$ and by above, for every $k \in \mathbb{Z}$ such that $y + KL > 2\gamma(p)$ there is a valid run from (s, x) to $(t, y + kL)$ in C^γ.

The case $P = \rho \times \sigma < 0$ is symmetric. □



Now, there are only $L(T)$ equivalence classes modulo $L(T)$. So consider nonlinear runs $\rho_1, \rho_2, \rho_3, \ldots$ ending in configurations $(t, y_1), (t, y_2), (t, y_3)$, etc. If some subruns ρ_i, ρ_j for $i < j$ finish in the same equivalence class modulo $L(T)$ (That is, we have $y_i \equiv y_j \bmod L(T)$) then, using the above lemma, there is a run from $\text{start}(\rho_i)$ to $\text{last}(\rho_j)$. And so we can skip all ρ_k for $i < k < j$.

First we define the notion of runs starting and finishing in the same configurations.

Definition 6.3.11. *Two run π, π' are equivalent, written $\pi \sim \pi'$, if* $\text{first}(\pi_1) = \text{first}(\pi')$ *and* $\text{last}(\pi) = \text{last}(\pi')$.

Then we can state the lemma on the rerouting of runs as follows.

Lemma 6.3.12. *Assume $\gamma(p) > 2n$. Given a run π in C^γ written as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_k \to \tau_k$ where π_i's are nonlinear runs and* $\text{counter}(\pi_i) > 2\gamma(p)$ *and* $\text{counter}(\text{start}(\pi_i)), \text{counter}(\text{last}(\pi_i)) \leq 3\gamma(p)$. *Then there is an equivalent run $\pi' \sim \pi$ such that π' factors as:*

$$\tau_0 \to \pi'_1 \to \tau_{f(1)} \to \pi'_2 \to \tau_{f(2)} \to \ldots \to \pi'_l \to \tau_{f(l)}$$

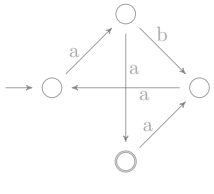
where

- $\pi'_i : \text{first}(\pi_{f(i-1)}) \to \text{last}(\pi_{f(i)}))$,
- $\text{counter}(\pi'_i) \geq \gamma(p)$,
- $f : [1 \ldots l] \to [1 \ldots k]$ *is a strictly increasing function,*
- $l \leq Dn^5$ *where D is the number of different types.*

Proof. Write the last configuration $\text{last}(\pi_i)$ of π as $\text{last}(\pi_i) = (t_i, y_i)$. Let L be the value from Lemma 6.3.10 associated with the type of π_1. According to Lemma 6.3.10, if there is i such that $t_i = t_1$ and $y_1 \equiv y_i \bmod L$ then there is a run π'_1 from $\text{first}(\pi_1)$ to (t_i, y_i). Set $f(1)$ to be the largest such i.

Then consider $\pi_{f(1)+1}$ and let L' be the value from Lemma 6.3.10 associated with its type. Now, if there is i' such that $t_{i'} = t_{f(1)+1}$ and $y_{f(1)+1} \equiv y_{i'} \bmod L'$ then, by Lemma 6.3.10, there is a run π'_2 from $\text{first}(\pi_{f(1)+1})$ to $(t_{i'}, y_{i'})$. Set $f(2)$ to be the largest such i'.

Repeat this process, until eventually we set $f(l) = k$ for some l. Now, for each type T and each state t the process is repeated at most $L(T)$ times (once for each equivalence class). Hence, $l \leq Dn \max_t L(t) \leq Dn^5$. □



6.3.9 Factoring of Runs

We now combine the above results and show that any run in C^γ has always an equivalent factoring of the desired form (6.1) described on page 144.

Theorem 6.3.13. *Assume $\gamma(p) > 2n$. Let π be a run in C^γ. Then there are $K, L \in \mathbb{N}$ depending only on C (and not on γ) such that π can be written as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ where $t \leq L$ and $\mathrm{counter}(\tau_i) \leq K\gamma(p)$ and $\gamma(p) < \mathrm{counter}(\pi_i)$ and $2\gamma(p) < \mathrm{counter}(\mathrm{start}(\pi_i)), \mathrm{counter}(\mathrm{last}(\pi_i)) \leq 3\gamma(p)$ for every i.*

Proof. Let π be an accepting run in C^γ and write $\pi = \sigma_0 \to \pi_1 \to \sigma_1 \to \pi_2 \to \sigma_3 \to \ldots \to \pi_k \to \sigma_k$ such that $\mathrm{counter}(\sigma_i) \leq 2\gamma(p) < \mathrm{counter}(\pi_i)$ for every i. Since every transition of C increases the counter by at most $\gamma(p)$, we have $2\gamma(p) < \mathrm{counter}(\mathrm{first}(\pi_i)) \leq 3\gamma(p)$.

Let Π be the collection of all nonlinear subruns π_i. Applying Lemma 6.3.12 to π gives us a split: $\pi = \tau'_0 \to \pi'_1 \to \tau'_1 \to \ldots \to \pi'_t \to \tau'_t$ such that $\mathrm{counter}(\pi'_i) \geq \gamma(p)$ and each τ'_i is a concatenation of some consecutive τ_j's and π_k's where $\pi_k \notin \Pi$. Further, $t \leq 3Dn^5$.

Thus, τ'_i can be written as $\tau'_i = \rho_1 \ldots \rho_r$ such that each ρ_i equals either (i) t_j for some j or (ii) π_j for some linear π_j or (iii) π_j for some nonpumpable π_j.

In each case, we can bound $\mathrm{counter}(\rho_i)$ as follows.

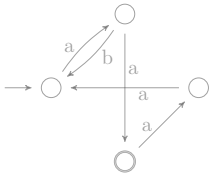
- If $\rho_i = \tau_j$ for some j then $\mathrm{counter}(\tau'_i) = \mathrm{counter}(\tau_j) \leq 2\gamma(p)$.
- If $\rho_i = \pi_j$ for some $\pi_j \notin \Pi$ and π_j is linear then, by Lemma 6.3.7, $\mathrm{counter}(\tau'_i) \leq \mathrm{counter}(\pi_j) = (3+n)\gamma(p)$.
- If $\rho_i = \pi_j$ for some $\pi_j \notin \Pi$ and π_j is not pumpable then, by Lemma 6.3.9, $\mathrm{counter}(\tau'_i) \leq \mathrm{counter}(\pi_j) \leq \max(x, y) + V\gamma(p)$.

So take K to be the maximum of the constants appearing above. □

6.3.10 Decision Procedures

We now show how to use the factoring of runs from Theorem 6.3.13 to decide the halting problem for C. Recall that the above results assume that $\gamma(p) > 2n$ to ensure that the notion of positive and negative loops is well defined. Thus, the first step in the decision procedure is to instantiate p at $p = 0, \ldots, 2n$ and check the existence of an accepting run (e.g., using Lemma 2.7.1) in the resulting nonparametric machines.

Suppose that for no value of $p = 0, \ldots, 2n$ we found an accepting run. Then we can assume that $\gamma(p) > 2n$. We now give ∃PAD formulae encoding the existence of an assignment γ and a factoring from Theorem 6.3.13 of an accepting run.



Theorem 6.3.14. *Given C and states $s, t \in C$, the $J(C, s, t) = \{(x, y, q) \mid (s, x) \to^* (t, y)$ in C^γ where $\gamma(p) = q$ and $q > 2n\}$ is* ∃PAD *definable.*

Proof. We encode the existence of accepting runs from Theorem 6.3.13. For runs τ_i, we use Theorem 6.2.1 to express reachability up to a given multiple of $\gamma(p)$. For runs π_i we use Lemma 6.3.1 and express them as runs in the parametric one-counter machine $C_{>p}$—reachability in which is ∃PAD definable (Lemma 2.7.1).

So, given l—the length of the factoring in Theorem 6.3.13 and states $u_0, u_1, \ldots, u_l$, $v_0, v_1, \ldots, v_l \in C$—the initial and final states of τ, respectively, consider the following formula:

$$J'(C, s, t, l, \overrightarrow{u}, \overrightarrow{v})(x, y, p) = \exists \overrightarrow{x}, \overrightarrow{y}. \quad \begin{array}{ll} \bigwedge_{i=1\ldots l} & G(C, s_i, t_i, K)(x_i, y_i, p) \\ \bigwedge_{i=1\ldots l} & \mathrm{Reach}(C_{>p}, t_{i-1}, s_i)(y_{i-1}, x_i, p) \\ \bigwedge_{i=1\ldots l} & (2p < x_i < 3p \ \wedge \ 2p < y_i < 3p) \\ \bigwedge & G(C, s_0, t_0)(x, y_0, p) \ \wedge \ y_l = y \\ \bigwedge & p > 2n \end{array}$$

The formula asserts the existence of a particular factoring from Theorem 6.3.13. Since there are only finitely many possible values for l, $\overrightarrow{u}$ and $\overrightarrow{v}$, the result follows as ∃PAD sets are closed under finite union. □

Since the satisfiability of ∃PAD-definable sets is decidable, we have:

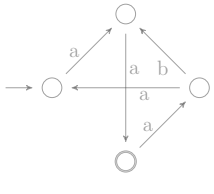
Theorem 6.3.15. *The halting problem is decidable for simple parametric bounded one-counter machines with a single parameter.*

6.3.10.1 Elimination of '$\equiv 0 \bmod c$' and '$+[0, p]$' Transitions

In the previous section, we showed decidability of the halting problem for simple parametric bounded one-counter machines. However, recall that our goal is to show decidability for parametric bounded one-counter machines as the halting problem for this class is equivalent (Theorem 5.2.10 and Theorem 5.2.7) to the halting problem for parametric timed automata with two parametric clocks. We now show how to extend our techniques to support '$\equiv 0 \bmod c$' and '$+[0, p]$' transitions in the case of a single parameter.

6.3.10.2 Elimination of '$\equiv 0 \bmod c$' Transitions

First suppose that C contains '$\equiv 0 \bmod c$' transitions but no '$+[0, p]$' transitions. Applying Theorem 6.3.14 to C_D's from Theorem 6.2.5 shows that the reachability for



machines with '$\equiv 0 \bmod c$' transitions is ∃PAD definable. Assuming $\gamma(p) > 2n$ we have:

Theorem 6.3.16. *Given C with '$\equiv 0 \bmod c$' transitions and states $s, t \in C$. The set $K(C, s, t) = \{(x, y, q) \in \mathbb{N}^3 \mid (s, x) \to^* (t, y)$ in C^γ where $\gamma(p) = q$ and $q > 2n\}$ is* ∃PAD *definable.*

Proof. According to Lemma 6.2.5 from page 140, it suffices to consider machines C_D for all conceivable D's. For a fixed $c \in \mathbb{N}$ the ∃PAD formula

$$\varphi_c(x, y) = (\exists q.c \cdot q + y = x) \wedge y < c$$

asserts that $x \equiv y \mod c$.

So let $\{c_1, \ldots, c_r\}$ be the set of constants appearing in '$\equiv 0 \bmod c$' transitions in C. Given $D \in \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$, we construct the formula $K'(C, s, t, D)(x, y, p)$ asserting that D is consistent with p and that $(s, x) \to^* (t, y)$ in C_D:

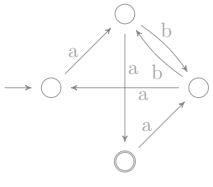
$$\begin{aligned} K'(C, s, t, D)(x, y, p) = \exists \overrightarrow{v}(r), \overrightarrow{w}(r) \quad . \quad & H(C_D, s \times v, t \times w)(x, y, p) \\ & \bigwedge_i (x \equiv v(i) \mod c_i) \\ & \bigwedge_i (y \equiv w(i) \mod c_i) \\ & \bigwedge_i (p \equiv D(i) \mod c_i) \\ & \wedge \ p > 2n \end{aligned}$$

Since ∃PAD definable sets are closed under finite union and there are only finitely many possible D's, we get the desired result. □

6.3.10.3 Elimination of '$+[0, p]$' Transitions

Now suppose that C is a parametric bounded one-counter machine C containing '$\equiv 0 \bmod c_i$' as well as '$+[0, p]$' transitions. Recall that in Lemma 6.2.6 we showed that if there is an accepting run in C^γ then there is an accepting run where in all but a finitely many cases the counter updates by '$+[0, p]$' transitions lie in the set $\{0, \ldots, R, \gamma(p) - R, \ldots, \gamma(p)\}$ where $R = \mathrm{lcm}(c_1, \ldots, c_r)$ is the least common multiple of all constants appearing in '$\equiv 0 \bmod c_i$' transitions.

The result, analogously to Theorem 6.2.7, can be directly used to show that the existence of an accepting run C^γ can be specified by a ∃PAD formula. Assuming $\gamma(p) > 2n$ we have:



Theorem 6.3.17. *Given C with '$\equiv 0 \bmod c$' and '$+[0,p]$' transitions and states $s,t \in C$ then the set $L(C,s,t) = \{(x,y,q) \in \mathbb{N}^3 \mid (s,x) \to^* (t,y)$ in C^γ where $\gamma(p) = q$ and $q > 2n\}$ is* $\exists$PAD *definable.*

Proof. Let Z be the machine obtained from C by replacing each '$+[0,p]$' transition by $2R+2$ transitions: $+0, +1, \ldots, +R, +p-R, \ldots, +p$. Then, by Lemma 6.2.6, we can assume that an accepting run π in C^γ can be factored as $\pi = \pi_0 \to \pi_1 \to \ldots \to \pi_v$ where each π is a run in Z^γ and there is a '$+[0,p]$' transition between π_i and π_{i+1}.

Given $v \leq nR$ and states $s = s_1, s_2, \ldots, s_v, t_1, \ldots, t_{v-1}, t_v = t$ the formula

$$L'(C,s,t,v,\overrightarrow{s},\overrightarrow{t})(x,y,p) = \exists \overrightarrow{x},\overrightarrow{y}. \quad \begin{array}{ll} & x_1 = x \ \wedge \ y_v = y \\ \bigwedge\limits_{i=1\ldots v} & K(Z,s_i,t_i)(x_i,y_i,p) \\ \bigwedge\limits_{i=1\ldots v-1} & 0 \leq y_{i+1} - x_i \leq p \\ \bigwedge\limits_{i=1\ldots v-1} & E(t_i, \text{`}+[0,p]', s_{i+1}) \\ \wedge & p > 2n \end{array}$$

asserts that such a factoring of length v exists where $\text{start}(\pi_i) = (s_i, x_i)$ and $\text{last}(\pi_i) = (t_i, y_i)$. The predicate $E(q, \text{`}+[0,p]', q')$ encodes that there is a '$+[0,p]$' transition between q and q' in C.

Notice there are only finitely many different values for $v' \leq nR$ and $\overrightarrow{s}$ and $\overrightarrow{t}$. Since $\exists$PAD sets are closed under finite union, the result follows. □

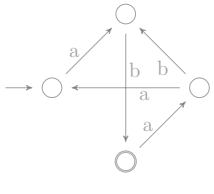
Since $\exists$PAD satisfiability is decidable, we have:

Theorem 6.3.18. *The halting problem for parametric bounded one-counter machines with a single parameter is decidable.*

6.3.10.4 Simple Programs

We now show how to use the developed theory to show decidability for simple programs—a model introduced by O. Ibarra *et al.* in 1990's [IJTW93]—the halting problem of which is still an open problem. A simple program is a simple parametric bounded one-counter machine (no '$+[0,p]$' or '$\equiv 0 \bmod c$' edges) N with the addition that the counter can become negative. For example, subtracting 5 when the counter equals 2 is valid and yields the counter equal to -3.

In case N has a single parameter p, we show that the halting problem for N is decidable. Suppose that there is an assignment γ and an accepting run in N^γ. Then consider a shortest accepting run π in N^γ. Split π into positive and negative subruns:



$\pi = \pi_1 \to \nu_1 \to \ldots \pi_k \to \nu_k$ for some k where $\text{counter}(\nu_i) \leq 0 \leq \text{counter}(\pi_i)$ for each i.

Note that Theorem 6.3.13 from the previous section applies to positive subruns. Hence, there are $K_1, L_1 \in \mathbb{N}$ such that all but L_1 subruns π_i satisfy, $\text{counter}(\pi_i) \leq K_1\gamma(p)$.

By multiplying the counter by -1, we can think of $-1 \cdot \nu_i$'s as runs in a simple parametric bounded one-counter machine. Similarly, there are $K_2, L_2 \in \mathbb{N}$ such that all but L_2 subruns ν_i satisfy, $\text{counter}(\nu_i) \geq -K_2\gamma(p)$.

Taking $K = \max(K_1, K_2)$ and $L = L_1 + L_2$ we get the following:

Theorem 6.3.19. *Let π be a run in N^γ. Then there are $K, L \in \mathbb{N}$ depending only on C (and not on γ) such that π can be written as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ where $t \leq L$ and $|\,\text{counter}(\tau_i)| \leq K\gamma(p)$ and either $\gamma(p) < \text{counter}(\pi_i)$ or $-\gamma(p) > \text{counter}(\pi_i)$ for every i. Further $2\gamma(p) < |\,\text{counter}(\text{start}(\pi_i))|, |\,\text{counter}(\text{last}(\pi_i))| \leq 3\gamma(p)$ for every i.*

Since the inverse $-1 \cdot \nu_i$ is a run in a parametric one-counter machine, similarly to Theorem 6.3.14, a factoring of the above form can be verified by a $\exists$PAD formula. Thus, we obtain:

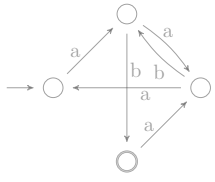
Theorem 6.3.20. *Given a simple program N and states $s, t \in C$, the set $H_{sp}(C, s, t) = \{(x, y, q) \mid (s, x) \to^* (t, y)$ in N^γ where $\gamma(p) = q\}$ is $\exists$PAD definable.*

Proof. Given N and states $s, t \in C$, the set $H_{sp}(C, s, t) = \{(x, y, q) \mid (s, x) \to^* (t, y)$ in N^γ where $\gamma(p) = q\}$ is $\exists$PAD definable.

Consider the machine $N_{<-p}$, which is obtained from N by removing all $\geq c, \geq p$ edges and replacing all $\leq c, \leq p$ edges by $+0$ edges. Finally, let M be obtained from N by multiplying all updates by -1. For example, edge $+p$ in $N_{<-p}$ becomes $-p$ in M.

Given l – the length of the factoring in Theorem 6.3.19 and states $u_0, u_1, \ldots, u_l$, $v_0, v_1, \ldots, v_l \in C$—the initial and final states of τ, respectively, consider the following formula:

$$H'_{sp}(C, s, t, l, \overrightarrow{u}, \overrightarrow{v})(x, y, p) = \exists \overrightarrow{x}, \overrightarrow{y}.\ \begin{array}{ll} \bigwedge\limits_{m=1\ldots l} & G(C, s_m, t_m, K)(x_m, y_m, p) \\ \bigwedge\limits_{i} & \text{Reach}(N_{>p}, t_{i-1}, s_i)(y_{i-1}, x_i, p) \\ \bigwedge\limits_{j} & \text{Reach}(M, t_{j-1}, s_j)(-y_{j-1}, -x_j, p) \\ \wedge & G(C, s_0, t_0)(x, y_0, p) \ \wedge\ y_l = y \\ \bigwedge\limits_{i} & (2p < x_i < 3p \ \wedge\ 2p < y_i < 3p) \\ \bigwedge\limits_{j} & (2p < -x_j < 3p \ \wedge\ 2p < -y_j < 3p) \end{array}$$



where the index i ranges over π_i with $\text{counter}(\pi_i) > \gamma(p)$ and the index j ranges over π_j with $\text{counter}(\pi_k) < -\gamma(p)$.

The formula asserts the existence of a particular factoring from Theorem 6.3.19. Since there are only finitely many possible values for $l, \overrightarrow{u}, \overrightarrow{v}$, the result follows as $\exists$PAD sets are closed under finite union. □

6.4 Discussion

In this chapter we studied the halting problem for parametric bounded one-counter machines. Building upon the previous work [Haa12], we showed decidability for the constant-update, "bounded" and the one-parameter fragment.

The resolution of the full general case is the main open problem of this thesis. In fact, the decidability is still open already for two parameters.

It is not clear whether the techniques developed in this chapter generalise to such multiparametric settings. We now outline the difficulties in lifting the argument.

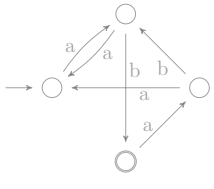
Recall from the proof (Theorem 6.3.14) for the one-parameter fragment that we partition an accepting run into subruns π_i's above a multiple of the parameter p and subruns τ_i's below the multiple of the parameter p. In multiparametric settings (assume parameters: $p_1 < \ldots < p_k$) we can analogously try to split a run into subrun ρ_i's above a multiple of p_k and subruns σ_i's below a multiple of p_k.

Now, runs ρ_i correspond to runs in an ordinary parametric one-counter machine $C_{>p_k}$ and thus are $\exists$PAD definable. The difficulty, however, lies in expressing the runs σ_i's. Even though we assume that $\text{counter}(\sigma_i) \leq c \cdot \gamma(p_k)$ for some $c \in \mathbb{N}$ we cannot a priori bound the ratio $\gamma(p_k)/\gamma(p_{k-1})$ independently of γ and hence bound the number of times '$+p_{k-1}$' edges can be taken before the counter exceeds $c \cdot \gamma(p_k)$.

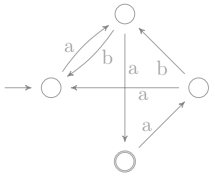
Also, it is not immediate how to appropriately lift to multiparametric settings the notion of linear and non linear pumpable runs as any such attempt seems to require knowing the relationship between p_j and all smaller $p_1, \ldots, p_{j-1}$. The property played a crucial role in bounding the number of π_i's and ρ_i's independently of an assignment and so even the length of the factoring might depend on γ in multiparametric settings.

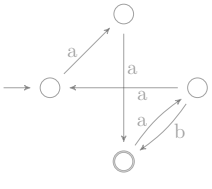
The decision procedures are of high complexity, on the other hand, the best lower bound for the halting problem for parametric bounded one-counter machines we are aware of (PSPACE) occurs already in nonparametric setting [FJ13]. Can the lower bound be improved by employing parameters?

Finally, note that the results presented in this chapter give $\exists$PAD characterisation of the reachability relation in the respective classes of parametric bounded one-counter



machines. Therefore, it is conceivable that the results can be used to solve other problems than plain reachability, e.g., model checking or the existence of a Büchi path in parametric bounded-one counter machines or the corresponding parametric timed automata.




a
a
a
a
b

Chapter 7

Epilogue

Many challenging and intriguing problems have been motivated or outright generated by formal verification over the years. In this thesis we just scratched the surface of automata and algorithmic problems originating in formal verification.

Already the problems covered in this thesis range over a wide spectrum. From very efficient AC^1 algorithms for the path-checking problem for Unary Temporal Logic to the problem of finding a satisfying assignments to parameters in parametric timed automata with decidability results with elementary complexity. See the Discussion section of individual chapters for a more thorough discussion on individual problems and for possible open problems and possibilities for future work. We now describe the main problems in this thesis.

7.1 Open Problems

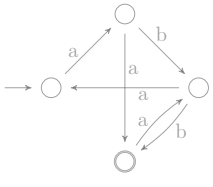
The main open problem left in this thesis is the decidability of the halting problem for parametric timed automata with two parametric clocks.

Open Problem 1. *Decidability of the following problem: Given a parametric timed automaton A with two parametric clocks, is there an assignment to parameters γ such that a final state is reachable in A^γ?*

The simplest open case is the following:

Open Problem 2. *Decidability of the following problem: Given a parametric timed automaton A with two clocks and using only two parameters, is there an assignment to parameters γ such a final state is reachable in A^γ?*

We believe that such a problem, and especially its general form without the restriction on the number of parameters, is best studied in terms of parametric



bounded one-counter machines. Analogously, in nonparametric settings a similar reduction [HOW12] proved crucial in resolving the precise complexity of reachability in two-clock timed automata [FJ13]. Using the reduction presented in this thesis (Theorem 5.2.10), such a parametric timed automaton A can be transformed into a parametric bounded one-counter machine C with two parameters.

Open Problem 3. *Decidability of the following problem: Given a parametric bounded one-counter machine C with two parameters, is there an assignment to parameters γ such that a final state is reachable in C^γ?*

Using the converse reduction (Theorem 5.2.7) such a one-counter machine C reduces to a parametric timed automaton with three parameters.[1] Thus, the second problem subsumes the third. It is not clear whether the problems are, in fact, interreducible.

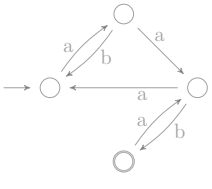
Notice that the second problem already subsumes decidability of the halting problem for parametric bounded one-counter machines with one parameter. The proof of the latter presented in this thesis is the culmination two thesis: the present one and [Haa12]. It is not clear whether such a decidability argument, proceeding via reduction to ordinary parametric one-counter machines and then to Presburger arithmetic with divisibility, generalises to more than one parameter. However, we hope that the machinery and techniques that have been developed will prove useful in attacking this problem and its general form:

Open Problem 4. *Decidability of the following problem: Given a parametric bounded one-counter machine C, is there an assignment to parameters γ such that a final state is reachable in C^γ?*

In Chapter 4 we studied the complexity of the path-checking problem for various temporal logics, the most prominent being LTL. Even though the lower and upper bound in this case are relatively close to each other (NC^1 vs. $\mathsf{AC}^1[\mathsf{logDCFL}]$), the lack of precise complexity for LTL path checking is dissatisfying. In particular, there has been no progress over the years on the trivial lower bound, which arises from LTL trivially subsuming propositional logic.

Open Problem 5. *Complexity of the following problem: Given a finite trace π and an LTL formula φ. Does π satisfy φ?*

[1] The new parameter represents the largest attained counter value.

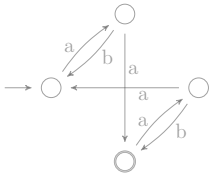


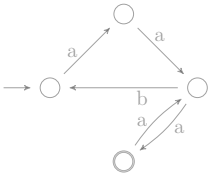
In Chapter 4 we studied the magnitude of completeness thresholds in bounded model checking. For regular languages we showed a strict linear vs. exponential dichotomy. However, for ω-regular properties, we were not able to fully prove a similar result. We conjecture that for ω-regular properties, a linear vs. quadratic vs. exponential trichotomy holds. The remaining open problem is as follows:

Open Problem 6. *Given a Büchi automaton B with more than quadratic completeness threshold, is the completeness threshold of B exponential?*

Another open problem from Chapter 3 is the complexity of calculating the completeness threshold for LTL properties. By first building an exponentially larger Büchi automaton and then applying the PSPACE decision procedure presented in this thesis an EXPSPACE algorithm is obtained. However, the best lower bound we are aware of is PSPACE-hard.

Open Problem 7. *Complexity of the following problem: Given an* LTL *formula φ, is the completeness threshold of φ linear?*




a
a
b
a
a

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[ABG+05] C. Artho, H. Barringer, A. Goldberg, K. Havelund, S. Khurshid, M. Lowry, C. Pasareanu, G. Rosu, K. Sen, W. Visser, and R. Washington. Combining test case generation and runtime verification. *Theoretical Computer Science*, 336(2-3), 2005.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, April 1994.

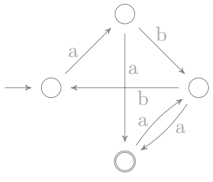
[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, January 1996.

[AHV93] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, 1993.

[AKRR11] Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded kolmogorov complexity in computational complexity theory. *J. Comput. Syst. Sci.*, 77(1):14–40, January 2011.

[BBE+10] T. Brázdil, V. Brozek, K. Etessami, A. Kucera, and D. Wojtczak. *One-counter Markov decision processes*, pages 863–874. Society for Industrial and Applied Mathematics, 2010.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.



[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207, 1999.

[BCGR92] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21:755–780, August 1992.

[Bel80] A.P. Bel'tyukov. Decidability of the universal theory of natural numbers with addition and divisibility. *Journal of Soviet Mathematics*, 14(5):1436–1444, 1980.

[BFS09] Florent Bouchy, Alain Finkel, and Arnaud Sangnier. Reachability in timed counter systems. *Electronic Notes in Theoretical Computer Science*, 239(0):167 – 178, 2009. Joint Proceedings of the 8th, 9th, and 10th International Workshops on Verification of Infinite-State Systems (INFINITY 2006, 2007, 2008).

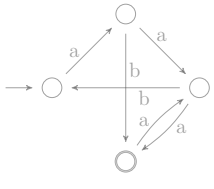
[BLMS99] D.A.M. Barrington, Chi-Jen Lu, P.B. Miltersen, and S. Skyum. On monotone planar circuits. In *Proceedings of Fourteenth Annual IEEE Conference on Computational Complexity, 1999.*, 1999.

[BO14a] Daniel Bundala and Joël Ouaknine. Advances in parametric real-time reasoning (to appear). In *Proceedings of MFCS*, volume ??? of *LNCS*. Springer-Verlag, 2014.

[BO14b] Daniel Bundala and Joël Ouaknine. On the complexity of temporal-logic path checking (to appear). In *Proceedings of ICALP*, volume 8573 of *LNCS*. Springer-Verlag, 2014.

[BOW12] Daniel Bundala, Joël Ouaknine, and James Worrell. On the magnitude of completeness thresholds in bounded model checking. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS '12, pages 155–164, Washington, DC, USA, 2012. IEEE Computer Society.

[Büc62] Julius R. Büchi. On a decision method in restricted second order arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science (LMPS'60)*, pages 1–11, June 1962.



[CD06] T. Chakraborty and S. Datta. One-input-face MPCVP is hard for L, but in LogDCFL. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*. 2006.

[CES08] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: Algorithmic verification and debugging. *CACM*, 52:75–84, 2008.

[Doy07] L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208 – 213, 2007.

[DS02] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84 – 103, 2002.

[FJ13] J. Fearnley and M. Jurdziński. Reachability in two-clock timed automata is PSPACE-Complete. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'13, 2013.

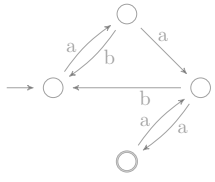
[FS04] B. Finkbeiner and H. Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24(2):101–127, October 2004.

[GGA04] Malay Ganai, Aarti Gupta, and Pranav Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *ICCAD*, pages 510–517, 2004.

[GHOW10] S. Göller, C. Haase, J. Ouaknine, and J. Worrell. Model checking succinct and parametric one-counter automata. In *ICALP'13*, volume 6199 of *Lecture Notes in Computer Science*. Springer, 2010.

[Gol77] L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9, July 1977.

[GPVW95] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In Piotr Dembinski and Marek Sredniawa, editors, *PSTV*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.



[Haa12] C. Haase. *On the Complexity of Model Checking Counter Automata*. PhD thesis, University of Oxford, 2012.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–, Washington, DC, USA, 1996. IEEE Computer Society.

[HKOW09] C. Haase, S. Kreutzer, J. Ouaknine, and J. Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR'09*, volume 5710 of *Lecture Notes in Computer Science*. Springer, 2009.

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*. Springer, 1992.

[HOW12] C. Haase, J. Ouaknine, and J. Worrell. On the relationship between reachability problems in timed and counter automata. In *RP*, volume 7550 of *Lecture Notes in Computer Science*. Springer, 2012.

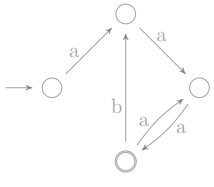
[HRSV01] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*. 2001.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[IJTW93] O. H. Ibarra, T. Jiang, N. Q. Trân, and H. Wang. New decidability results concerning two-way counter machines and applications. In *ICALP'93*, volume 700 of *Lecture Notes in Computer Science*. Springer, 1993.

[JLR13] A. Jovanovic, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. In *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, 2013.

[KF09] L. Kuhtz and B. Finkbeiner. LTL path checking is efficiently parallelizable. In *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming: Part II*, ICALP '09, 2009.



[KF12] L. Kuhtz and B. Finkbeiner. Efficient parallel path checking for linear-time temporal logic with past and bounds. *Logical Methods in Computer Science*, 8(4), 2012.

[KNSS02] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

[KOS+11] Daniel Kroening, Joel Ouaknine, Ofer Strichman, Thomas Wahl, and James Worrell. Linear completeness thresholds for bounded model checking. In *Proceedings of CAV*, volume 6806 of *LNCS*, pages 557–572. Springer, 2011.

[KR90] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*. 1990.

[Lip78] L. Lipshitz. The Diophantine Problem for Addition and Divisibility. *Transactions of The American Mathematical Society*, volume 235, 1978.

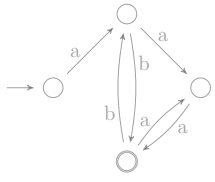
[Lip81] L. Lipshitz. Some remarks on the diophantine problem for addition and divisibility. *Proceedings of the Model Theory Meeting*, volume 33, 1981.

[LLT04] P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *Research Report LSV-04-16*, LSV, ENS de Cachan, 2004.

[LMS04] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer Berlin Heidelberg, 2004.

[LMS06] N. Limaye, M. Mahajan, and J. M. N. Sarma. Evaluating monotone circuits on cylinders, planes and tori. In *Symposium on Theoretical Aspects of Computer Science: STACS*, 2006.

[McM02] Kenneth McMillan. Applying SAT methods in unbounded symbolic model checking. In *CAV*, pages 250–264, 2002.



[McM03] Kenneth McMillan. Interpolation and SAT-based model checking. In *CAV*, pages 1–13. Springer, 2003.

[Mil00] J. S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*. 2000.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[MN04] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, 2004.

[MS03] N. Markey and Ph. Schnoebelen. Model checking a path (Preliminary report). In *Proceedings of the 14th International Conference on Concurrency Theory, Lecture Notes in Computer Science 2761*. Springer, 2003.

[OW03] Joël Ouaknine and James Worrell. Universality and language inclusion for open and closed timed automata. In *Proceedings of the 6th International Conference on Hybrid Systems: Computation and Control*, HSCC'03, pages 375–388, Berlin, Heidelberg, 2003. Springer-Verlag.

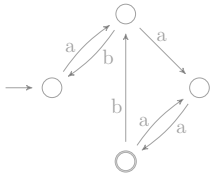
[Ric53] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[Spr00] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In Mathai Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1926 of *Lecture Notes in Computer Science*, pages 31–45. Springer Berlin Heidelberg, 2000.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD*, pages 108–125, 2000.

[Sud78] Ivan Hal Sudborough. On the tape complexity of deterministic context-free languages. *J. ACM*, 25(3):405–414, July 1978.

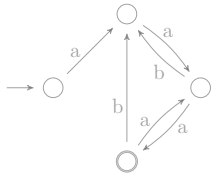


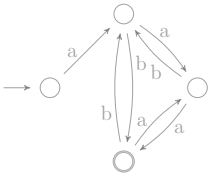
[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, November 1994.

[YS02] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proeedings of the 14th International Conference on Computer Aided Verification, volume 2404 of LNCS*. Springer, 2002.

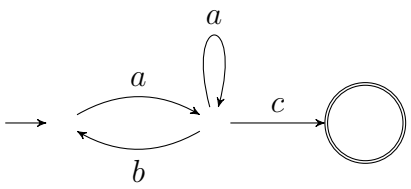



a
a
b
b
b
a
a

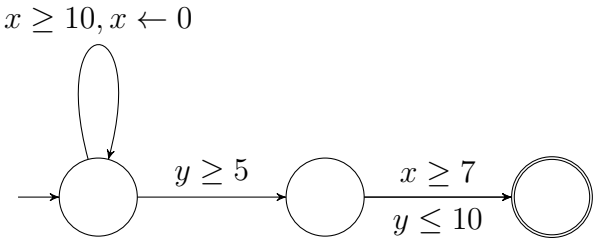
Appendix A

Reference Guide to Automata Models

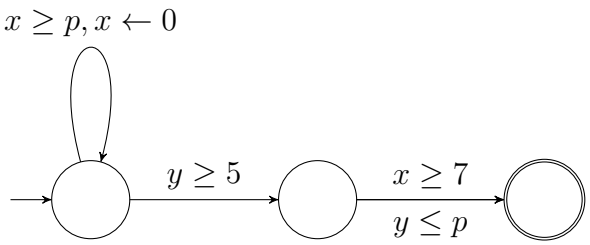
Finite Automaton over the alphabet Σ
Op: Σ

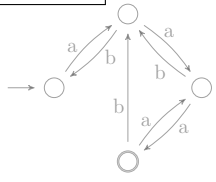


Timed Automaton with clocks C
Op: $2^C \times G(C)$

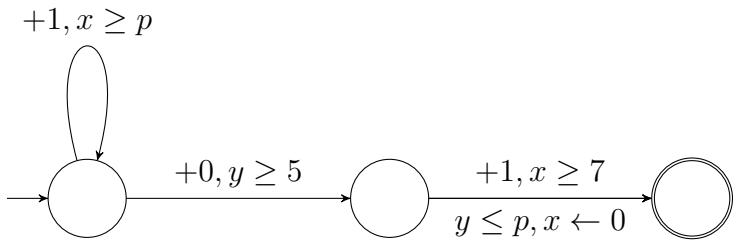


Parametric Timed Automaton with clocks C and parameters P
Op: $2^C \times G(P, C)$

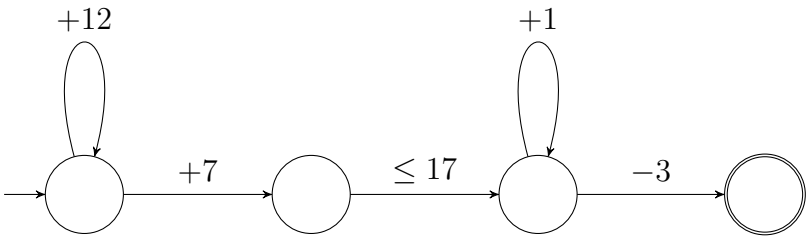




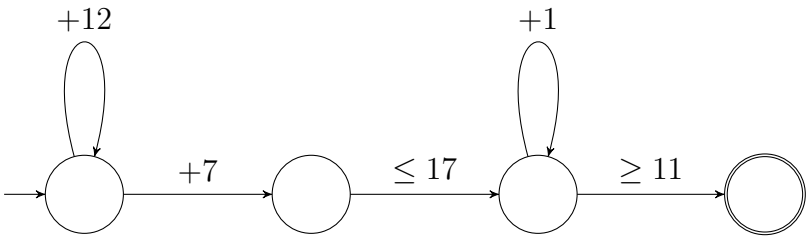
Parametric 0/1 Timed Automaton with clocks C and parameters P
Op: $\{+0, +1\} \times 2^C \times G(C, P)$



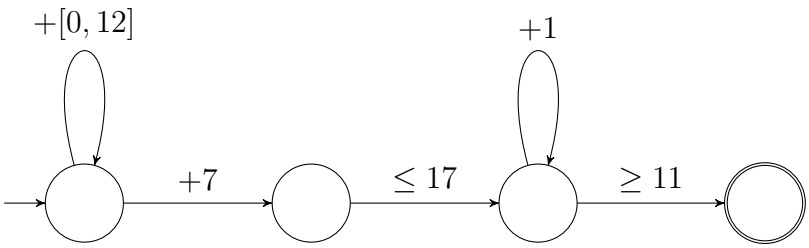
One-Counter Machine
Op: $\{+c, -c, =c, \geq c \ : \ c \in \mathbb{N}\}$

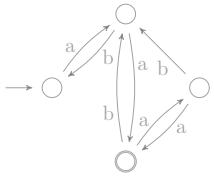


Simple Bounded One-Counter Machine
Op: $\{+c, -c, =c, \geq c, \leq c \ : \ c \in \mathbb{N}\}$



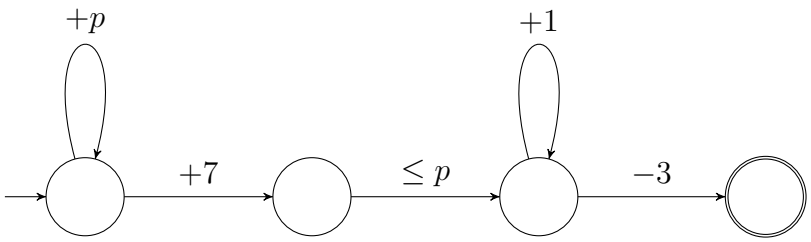
Bounded One-Counter Machine
Op: $\{+c, -c, =c, \geq c, \leq c, +[0, c], \equiv 0 \bmod c \ : \ c \in \mathbb{N}\}$





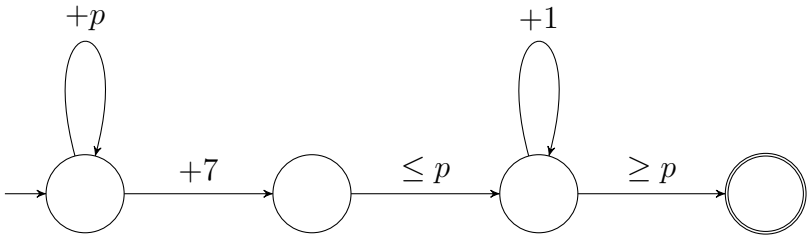
Parametric One-Counter Machine

Op: $\left\{ \begin{array}{ll} +c, -c, \geq c, = c & : \ c \in \mathbb{N} \\ +p, -p, \geq p, = p & : \ p \in P \end{array} \right\}$



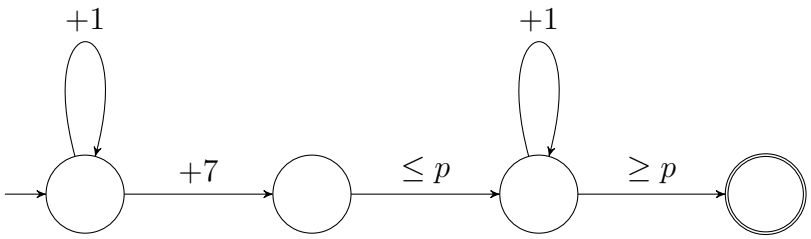
Simple Parametric Bounded One-Counter Machine

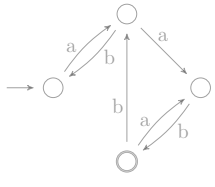
Op: $\left\{ \begin{array}{ll} +c, -c, \leq c, = c, \geq c & : \ c \in \mathbb{N} \\ +p, -p, \leq p, = p, \geq p & : \ p \in P \end{array} \right\}$



Parametric Bounded One-Counter Machine with Constant Updates

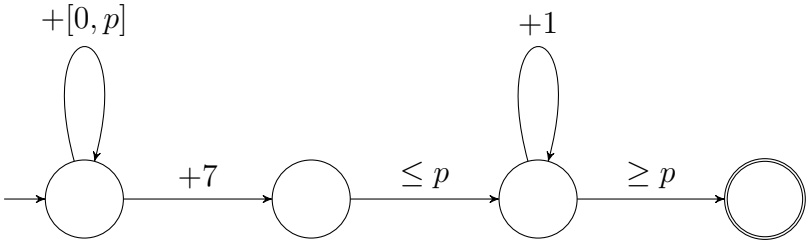
Op: $\left\{ \begin{array}{ll} +c, -c, \leq c, = c, \geq c & : \ c \in \mathbb{N} \\ \leq p, = p, \geq p & : \ p \in P \end{array} \right\}$





Parametric Bounded One-Counter Machine

Op: $\left\{ \begin{array}{ll} +c, -c, \leq c, = c, \geq c, +[0,c], \equiv 0 \text{mod } c & : \ c \in \mathbb{N} \\ +p, -p, \leq p, = p, \geq p, +[0,p] & : \ p \in P \end{array} \right\}$



Büchi Automaton over the alphabet Σ

Op: Σ