Synthesis and Alternating Automata over Real Time

Mark Jenkins

St John's College

University of Oxford

A dissertation submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy

Hilary Term 2012

Abstract

Alternating timed automata are a powerful extension of classical Alur-Dill timed automata that are closed under all Boolean operations. They have played a key role, among others, in providing verification algorithms for prominent specification formalisms such as Metric Temporal Logic. Unfortunately, when interpreted over an infinite dense time domain (such as the reals), alternating timed automata have an undecidable language emptiness problem. In this thesis we consider restrictions on this model that restore the decidability of the language emptiness problem.

We consider the restricted class of safety alternating timed automata, which can encode a corresponding Safety fragment of Metric Temporal Logic. This thesis connects these two formalisms with insertion channel machines, a model of faulty communication, and demonstrates that the three formalisms are interreducible. We thus prove a non-elementary lower bound for the language emptiness problem for 1-clock safety alternating timed automata and further obtain a new proof of decidability for this problem. Complementing the restriction to safety properties, we consider interpreting the automata over bounded dense time domains. We prove that the time-bounded language emptiness problem is decidable but non-elementary for unrestricted alternating timed automata.

The language emptiness problem for alternating timed automata is a special case of a much more general and abstract logical problem: Church's synthesis problem. Given a logical specification $S(I, O)$, Church's problem is to determine whether there exists an operator F that implements the specification in the sense that $S(I, F(I))$ holds for all inputs I. It is a classical result that the synthesis problem is decidable in the case that the specification and implementation are given in monadic second-order logic over the naturals. We prove that this decidability extends to MSO over the reals with order and furthermore to MSO over every fixed bounded interval of the reals with order and the $+1$ relation.

Acknowledgements

Firstly, I thank my supervisor, Professor James Worrell, for his guidance and encouragement throughout my research. I have greatly enjoyed working with him and am further grateful for his endless insight and patience.

I have been fortunate to collaborate with a number of brilliant researchers and in particular, I thank Joël Ouaknine, Alexander Rabinovich, Philippe Schnoebelen, Sylvain Schmitz, Stephan Kreutzer and Luke Ong for their questions, comments and advice.

My friends deserve great credit for making my time in Oxford so enjoyable and I particularly thank Tamsin, Ivan, Ellie, James and Fed for their help in proof-reading this dissertation. I would also like to thank my office-mates Thomas Gibson-Robinson and Ventsi Chonev for the many entertaining discussions we have shared. I thank my family for their immense support, care and love.

Finally, I am grateful for the financial support of the EPSRC, who funded this research through a Doctoral Training Award, as well as the Department of Computer Science and St John's College, who supported my conference travel.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Time-Bounded Automata 6

1.3 Time-Bounded Synthesis 9

1.4 Safety Properties 11

1.5 Authorship 14

2 Preliminaries **15**

2.1 Introduction 15

2.2 Timed Words 18

2.3 Timed Automata 19

2.3.1 Clocks and Constraints 19

2.3.2 Transitions and Automata 20

2.3.3 Semantics 21

2.3.4 Language Emptiness 22

2.3.5 Universality and Language Inclusion 25

2.4 Alternating Timed Automata 27

2.4.1 Syntax 28

2.4.2 Acceptance Game 29

2.4.3 Example . . . 30
2.4.4 Properties . . . 32
2.5 Metric Temporal Logic . . . 34
2.5.1 Syntax . . . 35
2.5.2 Pointwise Semantics . . . 35
2.5.3 Examples . . . 36
2.5.4 Properties . . . 37
2.6 Monadic Second-Order Logic . . . 38
2.7 Church's Problem . . . 40
2.7.1 McNaughton Games . . . 40
2.7.2 Graph Games . . . 41
2.7.3 Solving McNaughton Games . . . 44

3 Time-Bounded Alternating Timed Automata 47
3.1 Introduction . . . 47
3.2 Alternating Timed Automata . . . 50
3.3 Weak Monadic Second-Order Logic . . . 52
3.4 Real-Time McNaughton Games . . . 54
3.4.1 Defining a Game . . . 54
3.4.2 Solving the Parameter Problem . . . 55
3.5 Solving the Acceptance Game . . . 57
3.5.1 From Automata to Logic . . . 58
3.5.2 Eliminating the Metric . . . 60
3.5.3 A Series of McNaughton Games . . . 62
3.6 Hardness of Alternating Timed Automata . . . 63
3.6.1 Counter Machines . . . 64
3.6.2 Encoding Computation Histories . . . 65

3.6.3 Simulating Counter Machines . . . 68
3.7 Summary . . . 73

4 Real Time Uniformization **75**
4.1 Introduction . . . 75
4.2 Monadic Second-Order Logic . . . 78
4.3 Transforming Between Words and Signals . . . 79
4.4 Church's Problem with Parameters . . . 83
4.4.1 The Uniformization Problem . . . 84
4.4.2 From Signals to Words . . . 86
4.4.3 Stutter-Closed Uniformizers . . . 90
4.5 Uniformizing Metric Formulae . . . 95
4.5.1 Main Result . . . 96
4.6 Lower Bounds . . . 99
4.6.1 Star-Free Regular Expressions . . . 100
4.6.2 Encoding in $\mathsf{FO}(<, +1)$ . . . 102
4.7 Summary . . . 107

5 Safety Alternating Timed Automata **109**
5.1 Introduction . . . 109
5.2 Insertion Channel Machines . . . 112
5.2.1 Types of Channel Machine . . . 113
5.2.2 Machines with Errors . . . 114
5.2.3 Fairness . . . 115
5.3 Fair Termination for ICMs . . . 116
5.3.1 PSPACE Hardness . . . 116
5.3.2 Decidability in PSPACE . . . 120

5.4 Decidability of Fair Termination for ICMRs . . . 129
5.4.1 Removing Renaming . . . 129
5.4.2 Cycle Compression . . . 133
5.4.3 Higman's Lemma . . . 136
5.5 Hardness of Fair Termination for ICMOTs . . . 140
5.5.1 A Small Perfect Counter . . . 140
5.5.2 Inductive Sequences of Counters . . . 142
5.5.3 Large Perfect Counters . . . 146
5.6 Safety MTL . . . 148
5.6.1 Defining Safety MTL . . . 149
5.6.2 Encoding Fair Termination . . . 150
5.7 Safety Alternating Timed Automata . . . 153
5.7.1 Embedding Safety MTL . . . 155
5.7.2 Closing the Loop . . . 158
5.8 Summary . . . 165

6 Discussion **167**
6.1 Conclusions . . . 167
6.2 Related Work . . . 169
6.3 Future Work . . . 171

A Proofs from Section 4.3 **175**
A.1 Proof of Proposition 4.9 . . . 175
A.2 Proof of Proposition 4.10 . . . 179

B Proof of Proposition 5.24 **185**

Chapter 1

Introduction

1.1 Background

In safety-critical situations, one cannot tolerate failures of the computer systems used despite their increasing complexity. For example, a failure in the avionics of an aeroplane, the engine management system in a hydrogen car or the controller of a nuclear power station could be disastrous. Even for a simple hardware component such as a 256-bit adder, the number of possible combinations of the inputs is greater than the number of seconds since the big bang, so it is clear that exhaustive testing is impossible. Instead of relying on the creators of these systems to construct test suites which identify all interesting cases, we believe that *formal verification*, which aims to offer a mathematical proof that a system meets (some part of) its specification, should play an important role in ensuring the quality of safety-critical systems.

It is clear that with complex systems one cannot be expected to construct a rigorous proof of correctness by hand, so we consider only computer-assisted techniques for verification to be adequate. Moreover, we do not wish to consider techniques which require expert guidance such as *theorem proving*, preferring to focus only on automated verification

techniques.

For these techniques to work, it is crucial that the requirements for the system be formalised, typically as a specification S in an appropriate *temporal logic*. This specification will describe constraints on the desired behaviour of the system such as "resource R may only be accessed by one part of the system at a time" or "each request for access to R must eventually be granted". One problem is to ensure the *satisfiability* of S, that there exists at least some system which can meet it, but perhaps the most critical problem is that of *model checking*, which asks whether a particular system meets the given specification. If the system I is formalised as a structure in which this logic may be interpreted, the problem is then to determine whether $I \models S$, i.e. whether the system I is a model of the specification S.

The simplest model of time for these purposes is the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$, which allow us to capture the order of events in the system and discrete delays between them (for example as a number of clock ticks). Over this domain, Kamp's 1968 thesis [Kam68] contains a beautiful result which shows that linear-time temporal logic (LTL) is expressively equivalent to the first-order predicate logic $\mathsf{FO}(<)$. LTL is one of the most widely used specification logics thanks partly to this result and partly due to the existence of efficient algorithms for model checking which reduce formulae of LTL to equivalent *finite state automata* [VW86].

For a specification S, the language of its equivalent automaton $\mathcal{A}_S$ is the set of all possible behaviours allowed by the specification. A system is modelled as a *Kripke structure* which can be seen as a special type of automaton $\mathcal{A}_I$. The question of whether the system is a model of the specification is then the *language inclusion* problem of whether $L(\mathcal{A}_I) \subseteq L(\mathcal{A}_S)$ — whether every behaviour of the system is allowed by the specification.

This problem can be solved by reformulating it as a simpler problem of *language emptiness* — does there exist a behaviour which is possible in the system but forbidden by the

specification? Such a translation requires us to find an automaton which captures the complement of the specification (those behaviours which are forbidden), but this is possible because even automata which deal with infinite behaviours (such as Büchi or Muller automata) are closed under complementation; the problem is then whether $L(\mathcal{A}_I \cap \mathcal{A}_{\neg S}) = \emptyset$.

Another beautiful result is that these automata over infinite words are expressively equivalent with *monadic second-order* predicate logic $\mathsf{MSO}(<)$ over the naturals [Büc62] and since $\mathsf{MSO}(<)$ is strictly more expressive than $\mathsf{FO}(<)$, finite automata are strictly more expressive than LTL; one can define a restricted class of *counter-free automata* which are expressively equivalent with $\mathsf{FO}(<)$ and LTL (see the survey [DG08] for details).

These classical methods only consider the execution of the system as a sequence of discrete events and while the ordering of events allows one to specify bounded-response properties such as "every request must be acknowledged within 5 events", this model of time is often insufficient for real-world systems. Such systems may have *metric* specifications which require that particular amounts of time elapse between certain events, regardless of the number of events which are triggered in between, such as the brakes of a car being applied within 10ms of the pedal being depressed. For these systems, we need a *dense* model of time such as the non-negative real numbers $\mathbb{R}_+$ as well as some method of measuring the distance between events.

There has been a wide-ranging program of research devoted to "lifting the classical [theory] to real-time systems" [Tra95] and indeed the event-based model of execution can easily be lifted to describe real-time systems by annotating each event with a timestamp from $\mathbb{R}_+$. We naturally require that the timestamps in these *timed words* be strictly increasing and we further require that they be *non-Zeno* or *progressive*, i.e. that there not be an infinite number of events in any finite amount of time. Unfortunately, while successful in many regards, verification technology for real-time systems poses significant challenges.

One difficulty is that this real-time event-based semantics does not coincide with the

state-based approach of mapping each moment in time to the state of the system at that moment. This state-based semantics is naturally used with predicate logics as one interprets monadic predicates as subsets of $\mathbb{R}_+$. Many different temporal logics have been defined for real-time systems, yet none are canonical as none have been shown to be as expressive as $\mathsf{FO}(<, +1)$; there is no analogue of Kamp's theorem over $\langle \mathbb{R}_+, <, +1 \rangle$.

One of the most widely studied real-time temporal logics is *Metric Temporal Logic* (MTL), which extends LTL with time constraints on the temporal modalities [Koy90] and has different expressiveness depending on whether the event-based or state-based semantics is used [BCM05]. The satisfiability problem for MTL is undecidable in the state-based semantics [AFH96] or over infinite timed words [OW06a], leading to the definition of fragments such as MITL [AFH96] and Bounded MTL [BMOW07] whose satisfiability problems are decidable.

Alur and Dill's *timed automata* have proved a hugely successful formalism for real-time systems, with [AD94] currently the most cited paper in the journal *Theoretical Computer Science*. These automata augment classical nondeterministic finite automata over infinite words with clocks which can capture timing conditions by enabling or disabling transitions based on the current clock values. One of the reasons for the wide adoption of timed automata is that the problem of model checking formulae of the branching-time temporal logic TCTL against these automata is PSPACE-complete [ACD90], as is their language emptiness problem [AD94]. However, the dual problem of *universality* — given an automaton, does it accept every behaviour? — is undecidable for timed automata. This implies that not only is language inclusion also undecidable, but that timed automata cannot be closed under complementation. Moreover, this non-closure under complementation entails weaker expressiveness; timed automata are not as powerful as $\mathsf{MSO}(<, +1)$ over $\langle \mathbb{R}_+, <, +1 \rangle$.

There have been many attempts to restore the decidability of the language inclusion problem such as digitisation techniques [HMP92], restricting the number of clocks [OW04]

or defining a determinisable class of timed automata [AFH99]. The *event clock* automata of [AFH99] are closed under complementation but not under *projection*, so they are again not as expressive as $\mathsf{MSO}(<, +1)$ where projection corresponds to existential quantification. Alur-Dill timed automata, however, are closed under projection.

In one way, it is simple to extend timed automata to obtain a formalism which is closed under complementation by using the classical idea of *alternation*, which was first defined by Chandra, Kozen and Stockmeyer [CKS81] and applied to automata on infinite words by Miyano and Hayashi [MH84]. The nondeterminism of timed automata means that there may be a choice or disjunction of potential next states following a transition. Alternation generalises nondeterminism by allowing conjunction as well as disjunction in the transition function of the automaton — some transitions lead to more than one next state.

Under this model, the execution of an automaton is often represented as a tree where there is a single child state following each disjunctive transition and multiple children from conjunctive transitions; each level of the tree corresponds to a single letter of the input word. Ouaknine and Worrell [OW05] take this semantics in their definition of *alternating timed automata*, whilst Lasota and Walukiewicz [LW05] define instead an acceptance game between one player (Automaton) who resolves the disjunctive choices and another (Pathfinder) who resolves the conjunctive choices. In Chapter 2, we follow this acceptance game definition of the semantics of an alternating timed automaton as we exploit connections between this and other types of logical games in our results, but it can easily be seen that the two definitions are equivalent.

Allowing alternation may enable classical automata to be more succinct but it does not increase their expressive power [CKS81, MH84]. Alternating timed automata, however, are strictly more expressive than nondeterministic timed automata since they can easily be complemented by exchanging conjunctions and disjunctions and complementing the acceptance condition [LW05].

The price to be paid for this extra generality is that language emptiness becomes undecidable for alternating timed automata, a fact which follows immediately from the undecidability of universality for nondeterministic timed automata. Note that both universality and language inclusion reduce to language emptiness for alternating timed automata by the constructions shown above, thus both the satisfiability and model checking problems are instances of language emptiness. One therefore wishes to find a class of alternating timed automata which has a decidable language emptiness problem.

Seeking to work around Alur and Dill's proof of the undecidability of universality for nondeterministic timed automata, which requires two clocks, both Ouaknine and Worrell and Lasota and Walukiewicz concentrated on the class of 1-clock alternating timed automata over finite words and concluded that language emptiness was decidable for this class [LW05, OW05]. This class proved to be surprisingly flexible — for example, it subsumes MTL and thus Ouaknine and Worrell proved that the satisfiability problem for MTL over finite words is decidable [OW05]. In this dissertation we consider two other restrictions of alternating timed automata which have decidable language emptiness problems.

1.2 Time-Bounded Automata

The first restriction we consider is the idea of *time-bounded* executions of the system. Under this restriction one fixes a time bound N and requires that all events have timestamps less than N. This paradigm can be seen as analogous to the idea of *bounded model checking*, which simplifies classical verification problems by limiting the total number of events in an execution of the system. Unlike bounded model checking, however, time-bounded executions may have arbitrarily many events thanks to the density of time — but note that we do not consider cases in which there are infinitely many events before N due to our requirement that executions be non-Zeno.

This restriction to time-bounded executions of a system is still useful in many real-world contexts as systems are usually expected to run for only a finite duration before being shut down or restarted. For example, a run of a security protocol may have an *a priori* timeout after which it is abandoned. A control system for a car or aeroplane would also usually be started fresh at the commencement of a new journey. Note that in each case it would be simple to bound the maximum operating time of the system but it may be difficult to place a similar upper bound on the number of events which the system will encounter.

The paradigm of time-bounded verification has been explored by several researchers [RR94, BHKH05, KZ06] and was successfully applied by Ouaknine, Rabinovich and Worrell [ORW09] who proved that the time-bounded language inclusion problem for nondeterministic timed automata is 2-ExpSpace-complete.

The method of Ouaknine, Rabinovich and Worrell works by establishing decidability of a variety of logics (MTL, $\mathsf{FO}(<,+1)$ and $\mathsf{MSO}(<,+1)$) over bounded real time domains and then translating timed automata into an appropriate formula of the temporal logic MTL. This result required a natural extension of methods employed in the discrete time case [Büc60] to timed automata. Moreover, the authors established that $\mathsf{FO}(<,+1)$ and MTL are equally expressive over any fixed bounded time domain $[0, N)$ — this can be seen as a time-bounded extension of Kamp's theorem and demonstrates that bounded time domains have a cleaner logical theory than unbounded real time.

We therefore seek to apply the paradigm of time-bounded verification to alternating timed automata and consider the *time-bounded language emptiness* problem for an alternating timed automaton $\mathcal{A}$ and time bound N. The problem is to determine whether there exists any word w in $L(\mathcal{A})$ where all timestamps in w are less than N.

The first contribution of this dissertation is the following:

Theorem 3.10. *The time-bounded language emptiness problem for alternating timed automata is decidable.*

This result is consistent with the decidability of the time-bounded language inclusion problem for nondeterministic timed automata, but we remark that its complexity is much higher. Each time unit incurs an exponential blow-up in our algorithm for deciding time-bounded language emptiness (thus the algorithm is non-elementary in general). It is not possible to improve on this, however, as we have established a non-elementary hardness result using a reduction from bounded 2-counter machines [Min61].

In order to prove this result, we note that for a fixed alternating timed automaton $\mathcal{A}$, one can follow a method similar to [ORW09] to define a corresponding formula $\varphi_{\mathcal{A}}$ of $\mathsf{MSO}(<, +1)$, but instead of reducing to the satisfiability problem for $\mathsf{MSO}(<, +1)$ over $\langle [0, N), <, +1 \rangle$, we reduce to a *McNaughton game* [McN65] with an $\mathsf{MSO}(<, +1)$ winning condition.

The formula $\varphi_{\mathcal{A}}$ we construct has variables w, A and P and encodes the acceptance game of $\mathcal{A}$ on the time-bounded word w. The variables A and P describe the moves of Automaton and Pathfinder in the acceptance game for $\mathcal{A}$ on input w and $\varphi_{\mathcal{A}}$ holds if and only if A and P represent a play of this acceptance game which ended with Automaton winning. Note that we cannot simply existentially quantify out A and P as this would destroy the interplay of the acceptance game, but we do wish to quantify out w as the language emptiness problem asks whether there exists any w which is accepted by $\mathcal{A}$.

We therefore view $\varphi_{\mathcal{A}}$ as the winning condition in a type of McNaughton game in which interpretations of the variables are built by successive moves of two players. Our game, however, extends the classical notion of McNaughton games by considering real-time and by considering the w variable to be a *parameter* to the game which is known by both players at the start. Whilst the addition of parameters was considered in [Rab07a] and [HST09], the *parameter problem* addressed in our work is novel. This problem is to determine for which parameter values the first player has a winning strategy for the game — it allows us to solve the language emptiness problem as these parameter values correspond to words

accepted by the automaton.

1.3 Time-Bounded Synthesis

McNaughton games were originally conceived as a solution to Church's *synthesis problem*, which was first posed by Church over 50 years ago [Chu57]. The synthesis problem is to automatically construct an implementation of a system from its specification or determine that there can be no such implementation. A solution to this problem has potential for practical interest since not only does it subsume the satisfiability problem for detecting errors in the specification, it is also a step toward the goal of creating systems that are correct by design. Note, however, that synthesis is applicable to a narrower class of problems than model checking — one can hope an implementation satisfies a number of informal properties (such as usability or performance) as well as the formal properties verified with model checking, but a synthesised system would not necessarily be a good fit for any properties that are not formally specified. In some domains, though, it may be possible to specify all requirements of a system completely rigorously — for example, in the field of avionics control software all behaviours of the system must be well understood.

Church thought of the specification as a formula $S(I, O)$ of $\mathsf{MSO}(<)$ over $\langle \mathbb{N}, < \rangle$ which describes the desired relationship between the inputs I and outputs O of a sequential circuit. Such a circuit acts as a transducer on an infinite sequence of bits — it produces one bit of output after each bit of input, so it must act in a *causal* manner — the output bit $O(k)$ can depend only on bits $I(0), \ldots, I(k)$ of the input. Since Church was interested in the construction of sequential circuits, he further required that any solution be *finite-state*; a finite-state transducer of this sort is exactly a *Mealy automaton* [Mea55]. In light of Büchi's proof of the expressive equivalence of finite-state automata and monadic second-order logic [Büc62], we can equally think of a solution to the synthesis problem as an

$\mathsf{MSO}(<)$ formula that defines a *causal operator*. We say that such a formula *uniformizes* the specification S.

Under this logical formulation, it is natural to consider a real-time extension of the synthesis problem which changes the domain to the non-negative reals $\langle \mathbb{R}_+, <, +1 \rangle$ and allows specifications expressed in $\mathsf{MSO}(<, +1)$. Unfortunately, there are a number of complications which limit potential solutions to this extension of the synthesis problem.

Shelah famously proved that over $\langle \mathbb{R}_+, < \rangle$ satisfiability is undecidable, even for formulae of $\mathsf{MSO}(<)$ (which do not mention the $+1$ relation), if one allows quantification over arbitrary predicates [She75]. This result leads us to restrict consideration to *finitely-variable* predicates which change their values only finitely many times in any bounded interval and can be seen as analogous to our non-Zeno requirement for timed words. With this finite-variability restriction, decidability of $\mathsf{MSO}(<)$ satisfiability is restored [Rab02] and we show in Chapter 4 that the synthesis problem is also decidable:

Theorem 4.3. *Given an* $\mathsf{MSO}(<)$ *formula* $\varphi(\overline{X}, \overline{Y})$*, one can decide whether there is an* $\mathsf{MSO}(<)$*-definable causal operator that uniformizes* φ *over* $\langle \mathbb{R}_+, < \rangle$*. If such an operator exists, the algorithm computes a formula that represents the operator.*

When the $+1$ relation, which allows one to speak of metric properties such as "the distance between two points is one", is added to the logic to obtain $\mathsf{MSO}(<, +1)$, satisfiability again becomes undecidable over $\langle \mathbb{R}_+, <, +1 \rangle$ [HR04]. We thus return to our time-bounded model of real-time systems as we note that Ouaknine, Rabinovich and Worrell proved that $\mathsf{MSO}(<, +1)$ satisfiability is decidable over any fixed bounded interval of the reals $\langle [0, N), <, +1 \rangle$ [ORW09]. The second main contribution of this dissertation is that the synthesis problem is also decidable over such intervals:

Theorem 4.4. *Given an* $\mathsf{MSO}(<, +1)$ *formula* $\varphi(\overline{X}, \overline{Y})$ *and* $N \in \mathbb{N}$*, one can decide whether there is an* $\mathsf{MSO}(<, +1)$*-definable causal operator that uniformizes* φ *over the the interval*

$[0, N)$. *If such an operator exists, the algorithm computes a formula that represents the operator.*

Note that while our proofs of these two results crucially require the use of parameters in the synthesis problem (as in the extensions of McNaughton games considered in Chapter 3), these parameters are not our primary object of study.

We further prove that the time-bounded synthesis problem is much more complex than the satisfiability problem by demonstrating non-elementary hardness in the length of the interval, even for formulae of fixed quantifier depth (whose satisfiability problem is elementary), through a reduction from the language emptiness problem for star-free regular expressions [SM73].

1.4 Safety Properties

Properties that are useful in verification can often be thought of as falling into one of two intuitive categories — *safety* properties, which require that "something bad" never happen and *liveness* properties, which require that "something good" eventually happen [Lam77]. These categories were formalised by the semantic definitions of Alpern and Schneider [AS85] who proved that every property we might consider is the intersection of a safety property and a liveness property.

Many important verification properties can naturally be phrased as requiring that the system never enter an error state (or leave a set of normal states) and can therefore be expressed as safety properties. Note that *bounded-response* properties, which require that the system perform some task either within a specified number of state changes or within a time limit, are also safety properties as the "something bad" corresponds to the response bound being exceeded.

For alternating timed automata, the syntactic restriction of making all states accepting

is equivalent to this semantic requirement. We note that even safety alternating timed automata have an undecidable language emptiness problem in general, so we consider the 1-clock case only.

With the further restriction of *locality*, which requires the automata to reset their single clock every time they change state, Ouaknine and Worrell proved that language emptiness is decidable for local safety 1-clock alternating timed automata [OW06b]. We prove in Chapter 5 that this locality restriction is not required and obtain

Theorem 5.26. *The language emptiness problem is decidable for 1-clock safety alternating timed automata.*

Parys and Walukiewicz define in [PW09] a stronger class of *weak* alternating timed automata, where there are no transitions from accepting to non-accepting states and again only a single clock. Whilst the authors prove that language emptiness is decidable for this class, bounding the complexity of their technique remains an open problem.

We instead connect the class of 1-clock safety alternating timed automata to a syntactic subset of MTL known as *Safety* MTL using the auxiliary formalism of *insertion channel machines*. Properties definable in Safety MTL meet the semantic definition of safety, and Ouaknine and Worrell proved that Safety MTL satisfiability is decidable by reduction to the language emptiness problem for 1-clock local safety alternating timed automata [OW06b].

Insertion channel machines can be considered as a type of faulty Turing machine. Instead of a tape, these machines are equipped with an unbounded first-in first-out *channel* and can interact with it by reading a letter from its head or writing one to its tail. The machines are faulty in the sense that some of the read operations correspond to *insertion errors*, appearing to read any possible letter but leaving the channel unchanged. Ouaknine and Worrell proved that the reachability problem for these machines has non-primitive recursive complexity [OW05] and used this result to show the non-primitive recursive complexity of language emptiness for 1-clock alternating timed automata over finite words.

We consider a notion of *fairness* for these insertion channel machines which requires that every letter written to the channel be eventually read. This requirement serves to prohibit runs which suffer from infinitely many insertion errors whilst letters are waiting on the channel and can thus be seen as analogous to the non-Zeno requirement we place on timed words. We then consider the *fair termination* problem which asks, given an insertion channel machine $\mathcal{M}$, whether there exists a fair infinite computation of $\mathcal{M}$. We prove that this fair termination problem is PSPACE-complete.

We extend the definition of insertion channel machines by adding the operation of *renaming*, as studied in [BMOW07], which allows the machine to change the letters on the channel (for example by replacing all occurrences of a with b). These renaming operations can also be used to test the channel for any occurrence of certain letters or to delete all instances of a letter (see Section 5.2.1 for details).

We show that the addition of this renaming operation makes insertion channel machines expressive enough to encode 1-clock alternating timed automata and thus that the language emptiness problem for 1-clock safety alternating timed automata reduces to the fair termination problem for insertion channel machines with renaming. Moreover, we show that the fair termination problem for insertion channel machines with renaming can be reduced to the satisfiability problem for Safety MTL. Recalling Ouaknine and Worrell's reduction from the satisfiability problem for Safety MTL to the language emptiness problem for 1-clock safety alternating timed automata, we see that all three problems are interreducible.

This strong connection between the three formalisms of temporal logic, timed automata and channel machines allows us to analyse the complexity of all three by considering only the complexity of the fair termination problem. Renaming greatly increases the complexity of insertion channel machines and we recall the first few levels of the *extended Grzegorczyk hierarchy* of fast-growing functions in order to describe it [Grz53, LW70, CS08]. This hierarchy is defined by induction, with $F_0(n) \stackrel{\text{def}}{=} n+1$ and $F_{i+1}(n) \stackrel{\text{def}}{=} F_i^{n+1}(n)$, so $F_1(n) =$

$2n+1$, $F_2(n) = \Theta(n^2)$, $F_3(n) = \Theta\left(\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n}\right)$ and the Ackermann function is $F_\omega(n) \stackrel{\text{def}}{=} F_n(n)$. We prove that

Theorem 5.17. *The fair termination problem for insertion channel machines with renaming is* $\textsc{Space}(F_4(n))$*-hard.*

Whilst we further prove that the fair termination problem is decidable for insertion channel machines with renaming (Theorem 5.16), our method obtains no complexity bound — we conjecture that there exists a decision procedure for this problem whose complexity can be bounded by F_ω.

1.5 Authorship

Chapters 3 and 4 were joint work with Alexander Rabinovich, Joël Ouaknine and James Worrell, who framed the problem, suggested a research direction and provided feedback during the research process. The technical development is my own, but Alex, Joël and James assisted in polishing the presentation for publication in the proceedings of LICS [JORW10] and CSL [JORW11] respectively.

Chapter 5 was developed with James Worrell who again helped to frame the problem, suggested strategies to pursue and provided feedback on the progress of the research.

Chapter 2

Preliminaries

2.1 Introduction

We model the execution of a real-time system by a *timed word*, a linear sequence of discrete events equipped with increasing points in real-time. The principle of this model is that we view an execution of a system solely in terms of the events which occur; this is an event-based semantics. One could consider instead a state-based model which uses a *signal* to map each time point to a state of the system. As we explore in Chapter 4, there is a sense in which timed words are sufficient to model any reasonable signal by recording every change in the signal's value (i.e. every state change of the system) with an event.

One crucial feature of timed words is that events can occur after any real delay and this delay can be measured. This *density* of real time allows us to have an arbitrary number of events within any bounded interval and a requirement that some event must occur after one time unit allows us to relate two events regardless of the number of intervening events.

We are then able to explore natural *timed automaton* models of a system whose execution is modelled by a timed word as defined by Alur and Dill [AD94]. Whilst the *language emptiness* problem of whether such an automaton accepts any timed word is

PSPACE-complete, the problem of whether a timed automaton accepts every timed word is undecidable. This leads us to conclude that timed automata cannot be closed under effective complementation and to search for a more expressive model.

Lasota and Walukiewicz [LW05] and Ouaknine and Worrell [OW05] remedied this deficiency by adding alternation to Alur-Dill timed automata, leading to *alternating timed automata*, which are closed under all Boolean operations. For this class of automata, the language emptiness problem is interreducible with the universality problem (by considering the dual automaton), however the fact that this model extends timed automata means that language emptiness is undecidable in general. Since at least two clocks are required to carry out Alur and Dill's proof that universality is undecidable, Lasota and Walukiewicz concentrated on the case where the alternating timed automata are allowed only a single clock.

When only the finite words accepted by a 1-clock alternating timed automaton are considered, the language emptiness problem becomes decidable, although with non-primitive recursive complexity. This was independently discovered by Ouaknine and Worrell, who were more concerned with the connection between this 1-clock alternating timed automata model and the temporal logic MTL.

Metric Temporal Logic (MTL) is a linear-time temporal logic which extends the classical Linear Temporal Logic (LTL) by adding time constraints to the temporal modalities. The problem of whether there exists any finite word which satisfies a given MTL formula was shown to be satisfiable by reduction to the language emptiness problem for 1-clock alternating timed automata over finite words [OW05], though for infinite words the problem is undecidable in general [OW06a].

The translation from MTL to 1-clock alternating timed automata also works in the case of infinite words, so it is also profitable to examine restricted classes of 1-clock alternating timed automata which accept infinite words and whose language emptiness problem is

decidable. Ouaknine and Worrell [OW06b] proved that the class of *local safety* 1-clock alternating timed automata has a decidable language emptiness problem and identified the corresponding *safety* fragment of MTL while Parys and Walukiewicz [PW09] proved that the class of *weak* alternating timed automata has a decidable language emptiness problem and identified a corresponding fragment of an alternative real-time temporal logic TPTL.

Establishing decidability of these restricted logics is important as they remain strong enough to express many properties useful in verification such as *invariance* and *bounded response*, so automata-based algorithms which can verify that such properties hold even when the execution of the system considered is infinite are potentially useful.

The semantics of temporal logics such as MTL can be given in terms of an embedding into the "canonical" formalism of predicate logic. We therefore consider the monadic second-order predicate logic $\mathsf{MSO}(<, +1)$ which uses a $+1$ relation between points in order to capture the metric properties which are essential when considering real-time systems.

As well as the obvious question of *satisfiability* — whether there exists any interpretation of the variables X and Y such that a formula $\varphi(X, Y)$ holds — we also consider the more involved *synthesis problem*, first posed by Church [Chu57]. Church's synthesis problem is to transform an infinite stream of input bits $\mathbf{X} = x_0 x_1 \ldots$ into an infinite stream of output bits $\mathbf{Y} = y_0 y_1 \ldots$ such that the resulting streams satisfy some specification $\varphi(X, Y)$. Church saw each output bit y_i as the result of the computation of a sequential circuit on the input x_i, possibly storing information about the previous steps of computation. He therefore required that the transducer be *causal*, i.e. y_i depends only on x_j for $j \leq i$, and implementable by a *finite-state* circuit.

McNaughton's approach was to describe this synthesis problem over the naturals in terms of a two-player game in which $\mathbf{X}$ and $\mathbf{Y}$ are constructed by alternating moves of two players, one of whom wishes $\varphi(\mathbf{X}, \mathbf{Y})$ to hold and one of whom wishes the opposite.

The classical approach to determining the winner of a McNaughton game first trans-

lates the winning condition φ into a *Muller automaton* and then defines a *graph game* over the locations of this automaton. It was demonstrated by Büchi and Landweber that such a *Muller game* is determined and the winner need only use a finite-state winning strategy [BL69]. Since this proof is constructive, it also serves to demonstrate that either there exists a finite-state circuit which implements φ or else there is no causal transducer.

2.2 Timed Words

Timed words arise by augmenting a sequence of letters with a corresponding sequence of real-valued timestamps; we only consider non-negative reals to be valid timestamps and hereafter denote the set of non-negative reals by $\mathbb{R}_+$.

A *time sequence* $\tau = \tau_1\tau_2\ldots$ is a strictly monotonic sequence of non-negative reals, i.e. for all i, $\tau_i \in \mathbb{R}_+$ and $\tau_i < \tau_{i+1}$. A *Zeno* sequence is one which intuitively has infinite density — it has infinitely many elements but is bounded above. Motivated by the fact that real-world systems do not exhibit Zeno behaviour, we require that bounded time sequences are finite and that infinite time sequences are unbounded. This condition on infinite time sequences is also referred to as a *progress* requirement in the literature [AD94].

A *timed word* over an alphabet Σ is a pair (σ, τ) where $\sigma = \sigma_1\sigma_2\ldots$ is a word over Σ and τ is a time sequence. We require σ and τ to have the same length, so we also talk of the word as being a sequence of *timed events* (σ_i, τ_i). With each timed word we denote the associated untimed word over Σ by $\mathit{Untime}(\sigma, \tau) = \sigma$. We then define a timed language to be a set of timed words and note that associated with each timed language L is an equivalent untimed language denoted by $\mathit{Untime}(L)$.

2.3 Timed Automata

Timed automata were introduced by Alur and Dill in [AD94]. The definition presented below is slightly different to that of [AD94] in order to better illustrate the extension to alternating timed automata.

2.3.1 Clocks and Constraints

The key feature of timed automata is the addition of *clock variables* to classical automaton models which can control the transitions of the automata. The only ways in which an automaton can make use of its clock variables are to reset them as part of a transition or to compare them to integer constants and use the truth values of these comparisons to guard some of its transitions.

If the automaton is equipped with a set X of clock variables, we define the set $\Phi(X)$ of *clock constraints* φ inductively by

$$\varphi ::= x \bowtie k \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2$$

where $x \in X$, $k \in \mathbb{N}$ and $\bowtie \in \{\leq, <, >, \geq\}$. We note that we could have chosen to allow rational values for the constants or to allow comparisons between two clocks, but neither addition increases the expressive power of these automata [AD94].

A *clock valuation* ν is a mapping from X to $\mathbb{R}_+$ that describes the value of each clock variable. Each constraint φ defines a subset $[\varphi] \subseteq (\mathbb{R}_+)^X$ where it is true. We say that a valuation ν satisfies a constraint φ if and only if $\nu \in [\varphi]$.

In order to define the semantics of timed automata, we need a pair of subsidiary operations on clock valuations: given $t \in \mathbb{R}_+$ and a clock valuation ν, $\nu + t$ defines the clock valuation which maps x to $\nu(x) + t$ and is used to represent the evolution of time. We define $\mathbf{0}$ as the valuation that assigns 0 to every clock variable. The valuation $[Y := t]\nu$

assigns t to $x \in Y$ and $\nu(x)$ to $x \notin Y$, and is used to reset a subset of the clocks without changing the remaining ones.

2.3.2 Transitions and Automata

Each time the automaton makes a transition, it does so instantaneously and has the opportunity to reset a subset of its clock variables. In order to accommodate this we define the *moves* of the automaton as $Moves(X, L) \stackrel{\text{def}}{=} \mathcal{P}(X) \times L$, where L is the set of locations of the automaton.

We are then able to define the transition function of the automaton. We consider it as a partial function $\Delta : L \times \Sigma \times \Phi(X) \to \mathcal{P}(Moves(X, L))$ which maps triples consisting of a location, letter and clock constraint to a set of moves. Intuitively we intend the transition $\Delta(x, \sigma, \varphi)$ to be enabled only when the *guard* φ is satisfied. Since we associate a set of moves with each such transition, it is clear that we only require a single one to be enabled at any time to allow the full range of nondeterministic behaviours, so we make the following simplifying assumption:

Disjointness. For each location l, letter σ and pair of clock constraints φ and ψ such that $\Delta(l, \sigma, \varphi)$ and $\Delta(l, \sigma, \psi)$ are both defined, we require that $[\varphi] \cap [\psi] = \emptyset$.

We now have all the ingredients required for the main definition:

Definition 2.1. A *timed automaton* is a tuple $\mathcal{A} = (S, s_0, \Sigma, C, F, \delta)$, where S is a set of *locations*; $s_0 \in S$ is the *initial location*; $F \subseteq S$ is the subset of *accepting locations*; Σ is the *alphabet*; C is the set of *clock variables*; $\delta : S \times \Sigma \times \Phi(C) \to \mathcal{P}(Moves(C, S))$ is the transition function which satisfies the *Disjointness* condition.

2.3.3 Semantics

Given a timed automaton $\mathcal{A}$ as in Definition 2.1, we call the combination of its current location s and clock valuation ν its *state* (s, ν). We then define a *run* of automaton $\mathcal{A}$ on a timed word (σ, τ) as a sequence of states $(s_0, \mathbf{0}), (s_1, \nu_1), (s_2, \nu_2), \ldots$ which has a fixed initial state and represents a proper succession of states of the automaton. This means that for each i, there must exist a constraint φ_i such that the clock valuation $\nu_i' = \nu_i + (\tau_{i+1} - \tau_i)$, which represents ν_i after the time between the ith and $i+1$th events has passed, satisfies φ_i. We also require there to exist a move $(Y, s_{i+1}) \in \delta(s_i, \sigma_{i+1}, \varphi_i)$ such that the new clock valuation $\nu_{i+1} = [Y := 0]\nu_i'$ represents the modification of ν_i' by resetting the clocks in Y.

We are now in a position to define acceptance of a timed word by a timed automaton in both the case where the word is finite and the case that it is infinite. Although other acceptance conditions for infinite words are considered by Alur and Dill, we need only concern ourselves with Büchi acceptance.

Definition 2.2 (Finite Acceptance)**.** If the timed word (σ, τ) is finite, it has a last event (σ_n, τ_n) and any run of an automaton $\mathcal{A}$ on this word has a corresponding final state (s_n, ν_n). We say that a run is *accepting* if $s_n \in F$ and that $\mathcal{A}$ *accepts* (σ, τ) if there exists an accepting run of $\mathcal{A}$ on (σ, τ).

Definition 2.3 (Büchi Acceptance)**.** If the timed word (σ, τ) is infinite, we say that a run $(s_0, \mathbf{0}), (s_1, \nu_1), (s_2, \nu_2), \ldots$ of an automaton $\mathcal{A}$ is *accepting* if there exists $f \in F$ such that $s_i = f$ for infinitely many i. We say that $\mathcal{A}$ *accepts* (σ, τ) if there exists an accepting run of $\mathcal{A}$ on (σ, τ).

We define the language $L_f(\mathcal{A})$ to be the set of finite timed words accepted by $\mathcal{A}$ and the language $L_\omega(\mathcal{A})$ to be the set of infinite timed words accepted by $\mathcal{A}$. We omit the subscript and write simply $L(\mathcal{A})$ if it is clear from the context which of these two languages we refer to.

Example. The automaton $\mathcal{A}_{\text{delay}}$ (over the alphabet $\{a\}$) illustrated in Figure 2.1 accepts any word which has two timed events with a delay of exactly one time unit between them. We use a double-circle to denote accepting states and an incoming arrow to denote the initial state. The automaton has a single clock and begins each run in state $(s, 0)$. When a timed event is encountered (which must have letter a), it may either transition to location s while increasing the clock by the delay since the last event or transition to location t while resetting its single clock x to 0 (the choice is nondeterministic). When in location t, if it encounters an event when its clock does not have value 1, the only available transition is to return to location t with the clock increased according to the delay between events; if an event is encountered whose delay would increase the clock value to 1, it may alternatively transition to location u with the clock updated to 1. Hence the only way to exit the location t is to encounter an event exactly 1 time unit after entering. In location u, any encountered event causes the automaton to remain in the location u and increase the clock according to the delay since the last event. Since u is the only accepting location and it may only be reached by exiting the location t, the automaton only accepts timed words which contain a pair of a events separated by exactly one time unit.

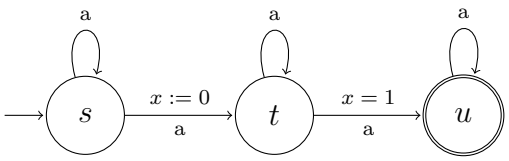


Figure 2.1: Automaton $\mathcal{A}_{\text{delay}}$

2.3.4 Language Emptiness

The *language emptiness* problem for timed automata asks, given a timed automaton $\mathcal{A}$, whether $L_\omega(\mathcal{A}) = \emptyset$; we refer to the related question of whether $L_f(\mathcal{A}) = \emptyset$ as the *finite word language emptiness problem*. Recall that for a timed word (σ, τ), $\mathit{Untime}(\sigma, \tau) = \sigma$,

i.e. *Untime* forgets the timestamps of the word. It is clear that $Untime(L_\omega(\mathcal{A})) = \emptyset$ if and only if $L_\omega(\mathcal{A}) = \emptyset$ (and similarly for finite words). Thus we seek a method of computing $Untime(L_\omega(\mathcal{A}))$ and can do this by abstracting the precise clock valuations into so-called *regions*, following the construction of [AD94]. The region automaton so constructed is exponential in the size of $\mathcal{A}$, but we can nondeterministically guess a path through it in space polynomial in the size of $\mathcal{A}$.

For the rest of this section, we fix a timed automaton $\mathcal{A} = (S, s_0, \Sigma, C, F, \delta)$ and let c_{max} be the largest integer appearing in a clock constraint in $\mathcal{A}$. For $t \in \mathbb{R}_+$, we say that $fract(t)$ is the fractional part of t and $\lfloor t \rfloor$ is the integer part of t, so $t = \lfloor t \rfloor + fract(t)$.

We then define an equivalence relation $\sim$ between clock valuations. $\nu \sim \nu'$ if and only if all of the following conditions hold:

- For all $x \in C$, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x) > c_{max}$ and $\nu(x') > c_{max}$.
- For all $x, y \in C$ with $\nu(x) \leq c_{max}$ and $\nu(y) \leq c_{max}$,
  $fract(\nu(x)) \leq fract(\nu(y))$ iff $fract(\nu'(x)) \leq fract(\nu'(y))$.
- For all $x \in C$ with $\nu(x) \leq c_{max}$, $fract(\nu(x)) = 0$ iff $fract(\nu'(x)) = 0$

The equivalence classes of $\sim$ are known as the *clock regions* of $\mathcal{A}$; we denote the clock region of ν by $[\nu]$. For a constraint φ and clock region α, observe that if $\nu, \nu' \in \alpha$ then $\nu \in [\varphi]$ if and only if $\nu' \in [\varphi]$, hence we say that α satisfies φ if and only if there exists $\nu \in \alpha$ such that $\nu \in [\varphi]$. The number of regions is clearly bounded and this bound is at most exponential in the size of $\mathcal{A}$ [AD94, Lemma 4.5]. The two key properties of clock regions are as follows: (i) for any constraint φ in $\mathcal{A}$, the set $[\varphi]$ of clock valuations satisfying φ is a disjoint union of clock regions; (ii) if $\nu, \nu' \in \alpha$ and there exist t and β such that $\nu + t \in \beta$ then there exists t' such that $\nu' + t' \in \beta$.

From a clock region α, we can compute its *time-successors*: regions α' such that for all $\nu \in \alpha$, there exists $t > 0 \in \mathbb{R}$ such that $\nu + t \in \alpha'$. In Figure 2.2, we illustrate the behaviour

for an automaton with two clocks x and y and maximum constant 2. The regions are the 9 horizontal line segments (e.g. from $(1,0)$ to $(1,1)$), the 9 vertical line segments, the 4 diagonal line segments, the 9 intersection points (e.g. $(1,0)$) and the 13 open spaces. For the illustrated point in the region with $1 < x < y < 2$, the possible time-successor regions are shown.

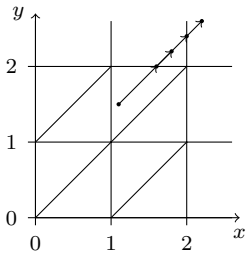


Figure 2.2: Illustration of time-successors of the $1 < x < y < 2$ region

Definition 2.4. The *region automaton* $R(\mathcal{A})$ has states of the form (s, α) where $s \in S$ and α is a clock region. Its initial state is $(s_0, [\mathbf{0}])$ and its accepting states are the set $\{(f, \alpha) \mid f \in F\}$. Its transition function is Δ, where there is a transition $(s', \alpha') \in \Delta((s, \alpha), \sigma)$ if and only if there exists $(Y, s') \in \delta(s, \sigma, \varphi)$ and a region α'' such that α'' is a time-successor of α, α'' satisfies φ and $\alpha' = [Y := 0]\alpha''$.

This construction is also known as a *time-abstract bisimulation quotient* of the state space of the timed automaton, i.e. $S \times (\mathbb{R}_+)^C$, because the states (s, ν) and (s, ν') are *bisimilar* in the sense that the same transitions are available from each if both $\nu, \nu' \in \alpha$ for some clock region α.

It is simple to show that when considered as a finite automaton, $R(\mathcal{A})$ exactly recognises $\mathit{Untime}(L_f(\mathcal{A}))$. Moreover, when considered as a Büchi automaton, $L_\omega(R(\mathcal{A}))$ is almost equal to $\mathit{Untime}(L_\omega(\mathcal{A}))$. The only difficulty is that $R(\mathcal{A})$ does not exclude infinite runs which would necessarily correspond to Zeno time sequences. For example, $R(\mathcal{A})$ could infinitely often take a transition with constraint $1 < x < 2$ without resetting the clock

x. This deficiency is simple to rectify, however — for each clock $x \in C$, we define $F_x = \{(s, \alpha) \mid \alpha \models [(x = 0) \vee (x > c_{max})]\}$ and modify the acceptance condition to require that some state from each F_x occurs infinitely often before a run is considered to be accepting.

We note that although $|R(\mathcal{A})|$ is exponential in $|\mathcal{A}|$, each state and its outgoing transitions can be represented in space polynomial in $|\mathcal{A}|$, so it is possible to check non-emptiness of $L(R(\mathcal{A}))$ using only space polynomial in $|\mathcal{A}|$ by guessing a path of the correct form through $|R(\mathcal{A})|$. Moreover, NPSPACE = PSPACE [Sav70] so we conclude that the language emptiness problem is decidable in PSPACE.

Note also that one can encode the computation of a *linear bounded automaton* M — a Turing machine which uses only the tape cells occupied by its input w — by a timed automaton $\mathcal{A}$ such that $L(\mathcal{A})$ is nonempty if and only if M halts on input w [AD94]. This construction implies that the language emptiness problem is PSPACE-hard and we conclude

Theorem 2.5 ([AD94])**.** *The language emptiness problem for timed automata is* PSPACE*-complete.*

Note that this problem becomes undecidable for several possible extensions of timed automata. Examples include allowing irrational constants in clock constraints [Mil00], allowing the addition of two clock values in constraints [AD94] and allowing *stopwatches* — clocks whose values are sometimes paused [HKPV95].

2.3.5 Universality and Language Inclusion

The *universality* problem for timed automata asks whether a given timed automaton $\mathcal{A}$ accepts all timed words. We show that this problem is undecidable by following Alur and Dill's construction of an encoding of the *recurrence* problem of 2-counter machines.

Definition 2.6. A *2-counter machine* M has a set of n locations and two counters C and D. We represent a configuration of such a machine by a triple $\langle i, c, d \rangle$ where $1 \leq i \leq n$,

$c \geq 0$ and $d \geq 0$. Each location corresponds to an instruction to increment or decrement one of the counters, or a jump instruction which is conditional on one of the counters being 0. All instructions have two possible next instructions; for the non-jump instructions, nondeterminism is used to decide which is actually visited.

The problem of deciding whether such a machine has a computation in which the initial location is visited infinitely often is undecidable, by reduction from an analogous recurring computation problem for a Turing machine with initially empty tape; the latter problem was shown to be undecidable in [HPS83].

To encode the computations of a 2-counter machine M with n instructions, we use timed words over the alphabet $\{l_1, \ldots, l_n, a, b\}$. One configuration is encoded in each time unit (the jth configuration is encoded in the interval $[j, j+1)$), with $\langle i, c, d\rangle$ encoded by the sequence $l_i a^c b^d$. A whole computation is encoded by the concatenation of the sequences which encode individual configurations.

Note that the density of time is crucial in this encoding as it allows for arbitrarily large counter values to be represented in a single time unit. We use the metric nature of time to enforce the requirement that the number of a symbols in successive time units differ by at most one, for example by requiring that the a symbols in each time unit (except possibly the last symbol) have corresponding a symbols one time unit later. We can hence define a timed language L_M which contains only those (infinite) timed words which encode a computation of M which visits the initial location infinitely often.

We construct a timed automaton $\mathcal{A}_M$ which accepts the complement of L_M; this accepts all timed words if and only if M has no computation which visits the initial location infinitely often. This automaton can be described as a disjunction of nondeterministic components which enumerate the ways in which a timed word can fail to encode such a computation of M; this enumeration is sketched in [AD94]. Hence we conclude the following:

Theorem 2.7 ([AD94])**.** *The universality problem is undecidable for timed automata.*

In Alur and Dill's proof of this theorem, infinite words and at least two clocks are needed, but the result was refined by Abdulla et al. [ADOW05] who clarified that with two clocks the universality problem remains undecidable even when finite words are considered, and showed that in the case of infinite words universality is undecidable even for timed automata with a single clock.

Note that this implies that timed automata cannot be effectively complemented in general as otherwise we could test whether a given automaton accepts every timed word by deciding whether its complement can accept any timed word. Indeed, it is shown in [Her98] that there is no timed automaton which accepts the complement of $L(\mathcal{A}_{\text{delay}})$, where $\mathcal{A}_{\text{delay}}$ is the timed automaton depicted in Figure 2.1.

The *language inclusion* problem asks, given timed automata $\mathcal{A}$ and $\mathcal{B}$, whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Note that one can easily construct a timed automaton $\mathcal{A}_{\text{univ}}$ which accepts every timed word. Hence the universality problem for $\mathcal{B}$ can be reduced to language inclusion as $\mathcal{B}$ can accept every timed word if and only if $L(\mathcal{A}_{\text{univ}}) \subseteq L(\mathcal{B})$. Thus we conclude:

Theorem 2.8 ([AD94])**.** *The language inclusion problem is undecidable for timed automata.*

2.4 Alternating Timed Automata

In this section, we define *alternating timed automata* by extending the definition of timed automata from the previous section. Alternation generalises the nondeterminism found in those automata by allowing *conjunctive* as well as *disjunctive* transitions. This means that we can no longer talk about runs of the automata in the same way. We could choose to follow [OW05] and consider *run trees* as a generalisation of runs as paths. In these structures, one move from each disjunction and every move from a conjunction is represented,

so we require all (linear) paths through the tree to be accepting in order for the tree to be accepting. However, for subsequent developments it is more convenient to follow the route of [LW05] and define instead an *acceptance game*. In this semantics, a run of the automaton is still a linear structure but we now have two players who resolve the disjunctive and conjunctive choices respectively. The similarity between this and other two-player games is one which we will exploit in Chapter 3.

2.4.1 Syntax

For a set A, we define the positive boolean combinations $\mathcal{B}^+(A)$ by the following grammar (where $a \in A$):

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \ .$$

Without loss of generality, we will assume that any such combination is in disjunctive normal form[1]. For a formula $\varphi = (a_1^1 \wedge \ldots \wedge a_{k_1}^1) \vee \ldots \vee (a_1^n \wedge \ldots \wedge a_{k_n}^n)$ of $\mathcal{B}^+(A)$, we say that $M \subseteq A$ is a *model* of φ if there exists i such that $\{a_1^i, \ldots, a_{k_i}^i\} \subseteq M$.

Recall from Definition 2.1 that the transition function δ of a nondeterministic timed automaton maps triples consisting of a location, letter and clock constraint to a set of moves, i.e. $\delta : L \times \Sigma \times \Phi(X) \to \mathcal{P}(Moves(X, L))$.

This function captures the nondeterminism of the automaton by having a set of moves from which the automaton can choose. For alternating timed automata, we instead allow positive Boolean combinations of moves so that conjunctive branching can be represented as well as disjunctive branching. This means that the transition function δ of an alternating timed automaton has type $\delta : L \times \Sigma \times \Phi(X) \to \mathcal{B}^+(Moves(X, L))$. Note that we retain the Disjointness assumption from the definition of timed automata for this transition function,

[1] We note that transforming a combination to disjunctive normal form may incur an exponential blow-up in size but this is inconsequential when the complexities of the decision procedures presented in this dissertation are considered.

i.e., that for each location l, letter σ and pair of clock constraints φ and ψ such that $\delta(l, \sigma, \varphi)$ and $\delta(l, \sigma, \psi)$ are both defined, we require that $[\varphi] \cap [\psi] = \emptyset$.

Definition 2.9. An *alternating timed automaton* is a tuple $\mathcal{A} = (S, s_0, \Sigma, C, F, \delta)$, where S is a set of *locations*; $s_0 \in S$ is the *initial location*; $F \subseteq S$ is the subset of *accepting locations*; Σ is the *alphabet*; C is the set of *clock variables*; $\delta : S \times \Sigma \times \Phi(C) \to \mathcal{B}^+(Moves(C, S))$ is the transition function which satisfies the *Disjointness* condition.

2.4.2 Acceptance Game

In order to decide whether an alternating timed automaton $\mathcal{A}$ accepts a finite or infinite timed word $w = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots$, we define an *acceptance game* $G(\mathcal{A}, w)$ where one letter of the word is read in each round; a position (s, ν, i) of the game is a triple which records the state of the automaton (its location s and clock valuation ν) as well as the number i of the current round. One player (Automaton) attempts to prove that $\mathcal{A}$ accepts w while the other (Pathfinder) tries to disprove it. The game is played as follows:

1. The game is played by two players called Automaton and Pathfinder.

2. Each play of the game has one round for each letter of w.

3. The play starts at the position $(s_0, \mathbf{0}, 0)$, with s_0 the initial state of $\mathcal{A}$ and $\mathbf{0}$ the valuation which assigns 0 to every clock.

4. In the $(i+1)$th round, suppose the play starts in (s_i, ν_i, i) and the letter $(\sigma_{i+1}, \tau_{i+1})$ is encountered. By the Disjointness assumption, there is at most one constraint φ_{i+1} such that $\nu'_i = \nu_i + (\tau_{i+1} - \tau_i)$ satisfies φ_{i+1} and $\delta(s_i, \sigma_{i+1}, \varphi_{i+1})$ is defined. If there is none, Pathfinder wins the game; otherwise we let $b = \delta(s_i, \sigma_{i+1}, \varphi_{i+1})$ and first require Automaton to choose a model M_{i+1} of b. In response, Pathfinder chooses a move from M_{i+1}. If he chooses the move (Y, s), the next round begins in the state $(s, [Y := 0]\nu'_i, i+1)$.

5. Case: w is finite. Let n be the length of w, then after the nth round, the game position is (s_n, ν_n, n) for some s_n. Automaton wins the game if $s_n \in F$, otherwise Pathfinder wins.

6. Case: w is infinite. In this case, Automaton wins the game if there exists some $f \in F$ such that for infinitely many i, the game position (s_i, ν_i, i) after round i has $s_i = f$, otherwise Pathfinder wins.

Call the sequence $(M_1, m_1), \ldots, (M_i, m_i)$ a *partial play*, where the M_j are the models chosen by Automaton and the m_j are the moves chosen by Pathfinder in the first i rounds. A *strategy* for Automaton in $G(\mathcal{A}, w)$ is a function whose inputs are a game position, a partial play up to that position, $\mathcal{A}$ and w; its output is a model M_{i+1} appropriate to that position. For example, in round $(i+1)$ the position would be (s_i, ν_i, i), the partial play would be $(M_1, m_1), \ldots, (M_i, m_i)$ and M_{i+1} would be a model of $\delta(s_i, \sigma_{i+1}, \varphi_{i+1})$. We say that a strategy is *winning* if by always choosing the move provided by this function, Automaton wins the game regardless of whatever Pathfinder plays.

We say that $\mathcal{A}$ accepts w if Automaton has a winning strategy in $G(\mathcal{A}, w)$. As for timed automata, we define the language $L_f(\mathcal{A})$ to be the set of finite timed words accepted by $\mathcal{A}$ and the language $L_\omega(\mathcal{A})$ to be the set of infinite timed words accepted by $\mathcal{A}$. We omit the subscript and write simply $L(\mathcal{A})$ if it is clear from the context which of these two languages we refer to.

2.4.3 Example

We now give an example of such an alternating timed automaton and for simplicity use an alphabet with only the single letter a. This automaton should accept all words in which the sequence of events in the interval $[0, 1)$ is identical to the sequence of events in the interval $[1, 2)$ up to the fractional parts of the associated timestamps. We can describe

this language as $L_{copy} = \{(a, t_1), \ldots, (a, t_{2n}) \mid t_{n+i} = t_i + 1 \text{ and } t_i < 1 \text{ for } 1 \leq i \leq n\}$. Since these automata are closed under intersection, we can break the job of specifying this language into two subsidiary automata and take their intersection to obtain $\mathcal{A}_{\text{copy}}$.

The first automaton will ensure that those timed events (a, t) seen in the first time unit have a corresponding event $(a, t+1)$ in the second time unit. This automaton $\mathcal{A}_{\text{forward}}$ has set of locations $\{q, r, s\}$ with q initial and $\{q, s\}$ accepting. It has a single clock c and its transition function is defined by

$$\delta(q, a, c < 1) = (\emptyset, q) \wedge (\{c\}, r) \qquad \delta(q, a, c \geq 1) = (\emptyset, q)$$

$$\delta(r, a, c = 1) = (\emptyset, s) \qquad \delta(r, a, c \neq 1) = (\emptyset, r)$$

$$\delta(s, a, c \geq 0) = (\emptyset, s)$$

This automaton is illustrated in Figure 2.3. As well as the double-circle to denote accepting states and incoming arrow to denote the initial state, we draw a line to join parts of a conjunctive transition (i.e. in the transition from q to both q and r); multiple transitions leaving the same state are otherwise assumed to be disjunctive.

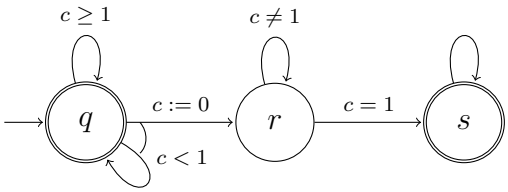


Figure 2.3: Automaton $\mathcal{A}_{\text{forward}}$

A run of $\mathcal{A}_{\text{forward}}$ begins in location q and whenever an a-event occurs in the first time unit (when $c < 1$, the clock starting at 0 at time 0) it moves either to q (to consider further events) or to r, resetting the clock c as it does so, at the Pathfinder's choice. It can only leave the non-accepting state r by making a transition to s exactly when $c = 1$, which is one time unit after seeing the a event which triggered the transition to r.

The second automaton $\mathcal{A}_{\text{back}}$, illustrated in Figure 2.4, must ensure that for every timed event (a,t) event in the second time unit, there is a corresponding event $(a,t-1)$ in the first time unit. Since the automaton cannot look back in time, we reformulate this condition by saying that for every successive pair of events (a,t_1), (a,t_2) in the first time unit, there is no event (a,t_3) with $t_1+1 < t_3 < t_2+1$. This automaton has set of locations $\{q, r_1, r_2, s_1, s_2, v\}$ with q initial and $\{s_1, s_2\}$ accepting. It has two clocks c and d and its transition function is defined by

$$\begin{aligned}
\delta(q,a,c<1) &= (\{c\},r_1) \wedge (\emptyset,q)) \vee (\{c\},r_2) & \delta(q,a,c\geq 1) &= (\emptyset,q)\\
\delta(r_1,a,d<1) &= (\{d\},s_1) & \delta(r_1,a,d\geq 1) &= (\emptyset,v)\\
\delta(r_2,a,d<1) &= (\emptyset,v) & \delta(r_2,a,d\geq 1\wedge c=1) &= (\emptyset,s_2)\\
\delta(r_2,a,d\geq 1\wedge c\neq 1) &= (\emptyset,r_2) & \delta(v,a,c\geq 0) &= (\emptyset,v)\\
\delta(s_1,a,c\leq 1\vee d\geq 1) &= (\emptyset,s_1) & \delta(s_1,a,c>1\wedge d<1) &= (\emptyset,v)\\
\delta(s_2,a,c\geq 0) &= (\emptyset,v)
\end{aligned}$$

A run of this automaton begins in state q and moves to state r_1 when it sees the first event in a pair, resetting the clock c as it does, and then to s_1 when it sees the second event in a pair, resetting the clock d as it does. The condition $t_1+1 < t_3 < t_2+1$ now corresponds to seeing an event between $c=1$ and $d=1$, so these events move the automaton to the non-accepting state v which it is unable to leave. The states s_2 and r_2 account for the last event seen before the end of the first time unit, which naturally does not have a succeeding event to pair it with.

2.4.4 Properties

In this section we review the properties of alternating timed automata, as detailed in the papers of Ouaknine and Worrell [OW05] and Lasota and Walukiewicz [LW05].

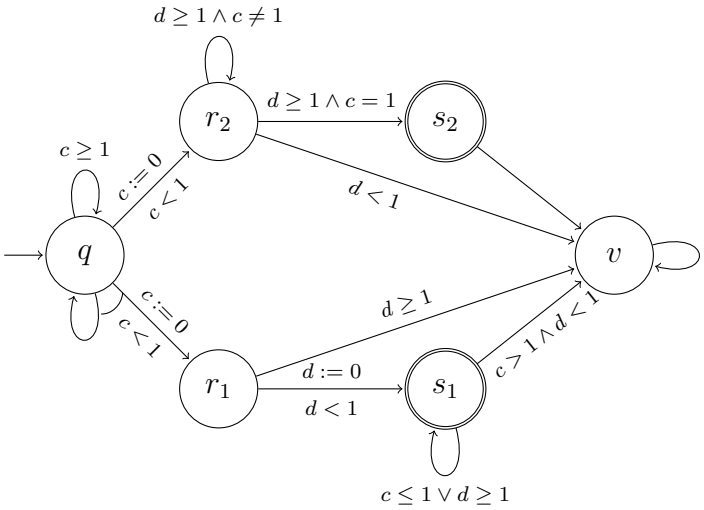


Figure 2.4: Automaton $\mathcal{A}_{\text{back}}$

Proposition 2.10 ([LW05])**.** *The class of languages accepted by alternating timed automata is effectively closed under the Boolean operations: union, intersection and complementation.*

Proof. Closure under conjunction and disjunction is straightforward as we allow any positive Boolean combinations of moves as a transition, so we can combine transitions where the automata overlap and still maintain the Disjointness condition.

In order to complement an alternating timed automaton $\mathcal{A}$, we first add an additional state $r \notin F$ and change the transition function to δ' which extends δ to a total function on $S \times \Sigma \times \Phi(C)$ by setting $\delta'(s, \sigma, \varphi) = r$ whenever $\delta(s, \sigma, \varphi)$ is undefined; note that these additions do not alter the language accepted by the automaton, but simply ensure that the acceptance game does not end early due to a transition being undefined. We then simply dualise the transition function by exchanging conjunctions with disjunctions in transitions; we also interchange accepting states with non-accepting states to obtain the complement automaton. □

Combining this result with Theorem 2.7 and the fact that alternating timed automata extend timed automata, we immediately obtain

Theorem 2.11. *The language emptiness problem for alternating timed automata is undecidable.*

Faced with this undecidability result, both [OW05] and [LW05] focus on two restrictions of this problem. Both consider only the *finite word language emptiness* problem which asks whether $L_f(\mathcal{A}) = \emptyset$, that is, whether any finite word is accepted by an alternating timed automaton $\mathcal{A}$. Moreover, both require that the alternating timed automaton $\mathcal{A}$ in question have only a single clock. In order to establish decidability of the problem with these restrictions, both [OW05] and [LW05] employ a time-abstract bisimulation quotient of the state space of the alternating timed automaton, as in Section 2.3.4. Since the semantics of alternating timed automata are more complex than those of nondeterministic timed automata, this quotient is no longer a finite object, but its decidability can be established by use of *well-quasi-orderings* (which are also used in Chapter 5).

Theorem 2.12. *The finite word language emptiness problem for 1-clock alternating timed automata is decidable.*

Note that both of these restrictions are crucial as Theorem 2.7 implies that finite word language emptiness is undecidable for alternating timed automata with at least two clocks and that (infinite word) language emptiness is undecidable even for 1-clock alternating timed automata.

2.5 Metric Temporal Logic

We now define *Metric Temporal Logic* (MTL) as a means of specifying properties of timed words. First defined by Koymans [Koy90], MTL extends the classical linear-time temporal logic LTL with time constraints on the temporal modalities.

Alur and Henzinger [AH93] subsequently considered instead extending LTL formulae with explicit clocks to cope with metric time and conjectured that the resulting logic TPTL

was strictly more expressive than MTL. This conjecture was settled positively over ten years later by Bouyer, Chevalier and Markey [BCM05]. Whilst it may seem that TPTL is more naturally related to timed automata thanks to the introduction of explicit clocks, in this dissertation we concentrate on MTL as it is possible to encode formulae of MTL as alternating timed automata in a particularly succinct way, namely by using only a single clock [OW05].

2.5.1 Syntax

Formulae of MTL are defined by the following grammar:

$$\varphi \stackrel{\text{def}}{=} \top \mid \bot \mid p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \bigcirc_I \varphi \mid \varphi_1 \, \mathbf{U}_I \, \varphi_2 \mid \varphi_1 \, \widetilde{\mathbf{U}}_I \, \varphi_2$$

where $p \in AP$ is an atomic proposition and I is an interval with endpoints in $\mathbb{N} \cup \{\infty\}$. $\top$ and $\bot$ are the usual *true* and *false* symbols and the *next* modality $\bigcirc_I \varphi$ requires that φ holds at the next timed event after a delay in I. The *until* modality $\varphi_1 \, \mathbf{U}_I \, \varphi_2$ requires that φ_2 holds at some event after a delay in I and φ_1 holds at every event before that while the *dual until* modality $\varphi_1 \, \widetilde{\mathbf{U}}_I \, \varphi_2$ is the dual of this condition in the sense that $\neg(\varphi_1 \, \mathbf{U}_I \, \varphi_2) \equiv (\neg\varphi_1) \, \widetilde{\mathbf{U}}_I \, (\neg\varphi_2)$.

We define the usual syntactic shorthands for the modalities *eventually*: $\Diamond_I \varphi \stackrel{\text{def}}{=} \top \, \mathbf{U}_I \, \varphi$, which requires that φ hold at some event after a delay in I and *always*: $\Box_I \varphi \stackrel{\text{def}}{=} \bot \, \widetilde{\mathbf{U}}_I \, \varphi$, which requires that φ holds at every event with a delay in I. We also use the propositional connective $\varphi_1 \rightarrow \varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2$.

2.5.2 Pointwise Semantics

We now define the *pointwise* semantics for MTL where we consider models of MTL formulae to be timed words $\rho = (\sigma, \tau)$ over an alphabet AP of atomic propositions. In this semantics, we define satisfaction of a formula only at positions i in the timed word and do not consider

the system at intervening times. The definition of satisfaction is by induction on the structure of the formula as follows:

$(\rho, i) \models \top$

$(\rho, i) \not\models \bot$

$(\rho, i) \models p$ iff $\sigma_i = p$

$(\rho, i) \models \varphi_1 \wedge \varphi_2$ iff $(\rho, i) \models \varphi_1$ and $(\rho, i) \models \varphi_2$

$(\rho, i) \models \varphi_1 \vee \varphi_2$ iff $(\rho, i) \models \varphi_1$ or $(\rho, i) \models \varphi_2$

$(\rho, i) \models \neg\varphi$ iff $(\rho, i) \not\models \varphi$

$(\rho, i) \models \bigcirc_I \varphi$ iff $(\rho, i+1) \models \varphi$ and $\tau_{i+1} - \tau_i \in I$

$(\rho, i) \models \varphi_1 \mathbf{U}_I \varphi_2$ iff $\exists j \geq i$ s.t. $(\rho, j) \models \varphi_2$ and $\tau_j - \tau_i \in I$ and $\forall i \leq k < j$, $(\rho, k) \models \varphi_1$

$(\rho, i) \models \varphi_1 \widetilde{\mathbf{U}}_I \varphi_2$ iff $\forall j \geq i$ s.t. $\tau_j - \tau_i \in I$, either $(\rho, j) \models \varphi_2$ or $\exists i \leq k < j$ s.t. $(\rho, k) \models \varphi_1$

Note that our semantics for the until modality $\varphi_1 \mathbf{U}_I \varphi_2$ correspond to those found in [OW06b] and are *strong* in the sense that they require φ_2 to eventually hold and *non-strict* in the sense that the formula is satisfied if φ_2 holds immediately and $0 \in I$.

We say that the timed word ρ satisfies φ if $(\rho, 1) \models \varphi$, define $L(\varphi)$ to be the set of timed words which satisfy φ and say that φ is satisfiable if $L(\varphi) \neq \emptyset$.

We say that a formula in which negation is applied only to atomic propositions is in *positive normal form*. Observing the definition of satisfaction above, we note that $\neg\bigcirc_I \varphi \equiv \top \widetilde{\mathbf{U}}_I (\neg\varphi)$ and that using this equivalence along with the duality of $\mathbf{U}_I$ and $\widetilde{\mathbf{U}}_I$ we can rewrite any formula of MTL into an equivalent formula in positive normal form.

2.5.3 Examples

We now present a number of examples of typical MTL formulae which express properties that may be useful for verification.

The formula $\Box\neg err$, which requires that *err* never holds, expresses an *invariance* property. The formula $\Box\Diamond ready$ requires that the system is always eventually *ready* and is an example of a *liveness* property. We can express *response* properties with a formula $\Box(req \rightarrow (\bigcirc_{\leq 2} ack \wedge \Diamond grant))$ which requires that every request is acknowledged by the next event, this acknowledgement must occur within 2 time units and the request must eventually be granted.

The formula $\varphi = \Box_{<1}(a \leftrightarrow \Diamond_{=1} a)$ requires that every a seen in the time unit $[0, 1)$ is matched by an a appearing one time unit later, as in the example of alternating timed automata from Section 2.4.3. Note, however, that this formula does not capture the language L_{copy} since there could be extra a events in the interval $[1, 2)$. For example, we show that the timed word $\rho = (a, 0.5), (a, 1.2), (a, 1.5)$ satisfies φ.

The formula φ is equivalent to $\bot \,\widetilde{\mathbf{U}}_{<1}\, \Big[(a \wedge (\top \,\mathbf{U}_{=1}\, a)) \vee (\neg a \wedge (\bot \,\widetilde{\mathbf{U}}_{=1}\, \neg a))\Big]$. Then

$$
\begin{aligned}
&(\rho, 1) \models \varphi \\
\text{iff } &(\rho, 1) \models (a \wedge (\top \,\mathbf{U}_{=1}\, a)) \vee (\neg a \wedge (\bot \,\widetilde{\mathbf{U}}_{=1}\, \neg a)) \text{ since only } \tau_1 < 1 \\
\text{iff } &(\rho, 1) \models (\top \,\mathbf{U}_{=1}\, a) \text{ since } (\rho, 1) \models a \\
\text{iff } &(\rho, 1) \models \top, (\rho, 2) \models \top \text{ and } (\rho, 3) \models a \text{ since } \tau_3 - \tau_1 = 1
\end{aligned}
$$

and the final set of conditions clearly holds.

2.5.4 Properties

We now consider the satisfiability problem for MTL with this semantics. By translating formulae of MTL to equivalent alternating timed automata with a single clock, Ouaknine and Worrell reduced the satisfiability problem for MTL to the language emptiness problem for 1-clock alternating timed automata [OW05]. Hence in conjunction with Theorem 2.12 they obtain the following:

Theorem 2.13 ([OW05])**.** *The* MTL *satisfiability problem over finite words is decidable.*

Over infinite words, however, this translation is unhelpful as the infinite word language emptiness problem is undecidable for alternating timed automata, even with a single clock. In fact, Ouaknine and Worrell were later able to encode another undecidable problem as an instance of infinite word MTL satisfiability, namely the *recurrent reachability* problem for a type of faulty Turing machines known as *insertion channel machines with occurrence testing* (ICMOTs).

Theorem 2.14 ([OW06a])**.** *The* MTL *satisfiability problem over infinite words is undecidable.*

In Chapter 5, we explore these ICMOTs further, show that the weaker problem of *fair termination* is decidable and exhibit a fragment of MTL whose satisfiability problem is interreducible with the fair termination problem.

2.6 Monadic Second-Order Logic

We consider *monadic second-order* logic $\mathsf{MSO}(<, +1)$ over a signature consisting of the binary relations $<$ and $+1$ and a countable family of monadic predicate names $P_0, P_1, \ldots$. The vocabulary of $\mathsf{MSO}(<, +1)$ also includes *first-order* variables $t_0, t_1, \ldots$ and monadic *second-order* variables $X_1, X_2, \ldots$. Atomic formulae are of the form $X(t)$, $P(t)$, $t_1 < t_2$, $+1(t_1, t_2)$ or $t_1 = t_2$. Well-formed formulae are obtained from atomic formulae using Boolean connectives, the first-order quantifiers $\exists t$ and $\forall t$, and the second-order quantifiers $\exists X$ and $\forall X$. We denote by $\mathsf{MSO}(<)$ the sub-language consisting of all formulae that do not mention the $+1$ relation. We use the abbreviation $\exists^{=1} t\, \varphi(t)$ to mean that there exists exactly one t such that $\varphi(t)$:

$$\exists^{=1} t\, \varphi(t) \stackrel{\text{def}}{=} \exists t(\varphi(t) \wedge \forall u(u \neq t \to \neg\varphi(u)))\,.$$

We also use $\exists^{\omega} t\, \varphi(t)$ to mean that there exist infinitely many t such that $\varphi(t)$:

$$\exists^{\omega} t\, \varphi(t) \stackrel{\text{def}}{=} \exists X\, [\forall t(X(t) \to (\varphi(t) \wedge \exists u(t < u \wedge X(u))) \wedge \exists t(X(t))] \ .$$

In this dissertation, we are interested in structures of the form $\mathcal{M} = \langle \mathbb{T}, <, +1, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ where $\mathbb{T}$ is an interval of non-negative reals with the usual order, $+1(x, y)$ holds if and only if $y = x + 1$, and $\mathbf{P}_1, \ldots, \mathbf{P}_m$ are subsets of $\mathbb{T}$ that interpret the monadic predicate names $P_1, \ldots, P_m$. (Generally we use boldface to denote interpretations of predicate names.) We omit the standard definition of what it means for a structure to satisfy a sentence. A formula $\varphi(P_1, \ldots, P_m, X_1, \ldots, X_n)$ with free second-order variables among $X_1, \ldots, X_n$ is interpreted in a structure $\langle \mathcal{M}, \mathbf{X}_1, \ldots, \mathbf{X}_n \rangle$ obtained by expanding $\mathcal{M}$ with interpretations of $X_1, \ldots, X_n$.

We also consider discrete structures for $\mathsf{MSO}(<)$ of the form $\langle \mathbb{T}, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$, where $\mathbb{T}$ is an initial segment of the natural numbers with the usual order and $\mathbf{P}_1, \ldots, \mathbf{P}_m$ are subsets of $\mathbb{T}$. Note that over these structures the addition of a $+1$ relation offers no additional expressive power, in contrast to the situation over the reals.

Example. We now give some examples of $\mathsf{MSO}(<, +1)$ formulae in the free variable A which we can think of as corresponding to a timed events. In order for this correspondence to be possible, A should hold only instantaneously i.e. for every t where $A(t)$ holds, there should exist $x < t$ and $u > t$ such that A does not hold anywhere in (x, t) or (t, u). We can express this requirement by the formula

$$\varphi_{instant} = \forall t \left[A(t) \to \left(\begin{array}{l} \exists u(t < u \wedge \forall v(t < v < u \to \neg A(v))) \wedge \\ \exists x(x < t \wedge \forall y(x < y < t \to \neg A(y))) \end{array} \right) \right]$$

We can then define a class of structures $\mathcal{W}$ using this requirement, so $\mathcal{M} \in \mathcal{W}$ if $\varphi_{instant}$ is true in $\mathcal{M}$.

Then the example timed automaton $\mathcal{A}_{\text{delay}}$ from Section 2.3.3 is equivalent over the

class of structures $\mathcal{W}$ to the formula

$$\varphi_{\text{delay}} \stackrel{\text{def}}{=} \exists t(A(t) \wedge \exists u(u = t + 1 \wedge A(u)))$$

which requires that there is some t for which A holds at t and $t + 1$.

The alternating timed automaton $\mathcal{A}_{\text{copy}}$ from Section 2.4.3 is equivalent over the class of structures $\mathcal{W}$ to the formula

$$\varphi_{\text{copy}} \stackrel{\text{def}}{=} \forall t\,[t < 1 \rightarrow (A(t) \leftrightarrow \exists u(u = t + 1 \wedge A(u)))]$$

which requires that for all $t < 1$, A holds at t if and only if A holds at $t + 1$.

2.7 Church's Problem

Consider an infinite sequence of bits $x = x_1, x_2, \ldots$ which is to be transformed one bit at a time into a sequence $y = y_1, y_2, \ldots$. The *synthesis problem* asks for us to derive from a specification of the desired relationship between the two sequences a method of implementing it, or to determine that there is no such method. When this problem was posed by Church [Chu57] more than 50 years ago, he asked that the implementation be a circuit and the specification be given in restricted arithmetic, but the more modern equivalents are known as *Mealy automata* [Mea55] and *monadic second-order logic* respectively. McNaughton [McN65] reformulated this problem as a game between two players, one of whom controls variables which correspond to the input and the other controlling variables corresponding to the output.

2.7.1 McNaughton Games

We now follow McNaughton's presentation of the classical synthesis problem as a logical game with an MSO winning condition and describe a solution to this game in Sections 2.7.2 and 2.7.3. Our subsequent developments do not depend on this classical solution in a

technical sense, but we include this material since our work generalises a number of the ideas presented below.

We consider the structure $\langle \mathbb{N}, < \rangle$ and let $\varphi(X, Y)$ be an $\mathsf{MSO}(<)$ formula, where X and Y are monadic predicate variables. The *McNaughton game* $\mathbb{G}(\varphi)$ is defined as follows:

1. The game is played by two players, called Player I and Player II.
2. Each play of the game has one round for each $n \in \mathbb{N}$.
3. In the ith round, Player I chooses $\mathbf{X}(i) \in \{0, 1\}$, then Player II chooses $\mathbf{Y}(i) \in \{0, 1\}$.
4. By the end of the play two interpretations $\mathbf{X}, \mathbf{Y} \subseteq \mathbb{N}$ of the predicates X and Y have been constructed. Player I wins if $\langle \mathbb{N}, <, \mathbf{X}, \mathbf{Y} \rangle \models \varphi(X, Y)$, otherwise Player II wins.

A *partial play* here is a finite sequence of choices $(x_0, y_0), \ldots, (x_{i-1}, y_{i-1})$ of Player I and Player II and a *strategy* for Player I is a function whose input is a partial play (up to some round i) and whose output is simply a choice in $\{0, 1\}$ for $\mathbf{X}(i)$. A strategy for Player I is *winning* if by always choosing the move it provides, Player I wins any play regardless of the moves made by Player II. We classify a strategy as *finite-state* if it can be implemented by a finite-state transducer; this is exactly the Mealy automaton mentioned earlier.

The *synthesis problem* for $\mathsf{MSO}(<)$ asks, given a formula $\varphi \in \mathsf{MSO}(<)$ whether Player I has a finite-state winning strategy in $\mathbb{G}(\varphi)$ and moreover to construct such a strategy if one exists.

In order to solve this problem, we introduce the notion of *graph games* and follow the presentation of Thomas [Tho08] in doing so.

2.7.2 Graph Games

A *graph game* has the form $G = (Q, Q_I, E)$ where Q is the set of vertices, $Q_I \subseteq Q$ is a set of Player-I vertices and $E \subseteq Q \times Q$ is the set of edges. We define here only 2-player graph

games and thus designate the remaining vertices Player-II vertices by setting $Q_{II} \stackrel{\text{def}}{=} Q \setminus Q_I$. Moreover we consider only games where the set of vertices Q is *finite*.

A *play* over G from q is a (finite or infinite) sequence $\rho = q_0 q_1 q_2 \ldots$ with $q_0 = q$ and $(q_i, q_{i+1}) \in E$ for $i \geq 0$. Player I chooses the next vertex from a vertex in Q_I and Player II chooses the next vertex from a vertex in Q_{II}. A *partial play* is a finite play and a *strategy* for Player I (respectively Player II) maps each partial play which ends in a vertex q_m in Q_I (respectively Q_{II}) to a new vertex q_{m+1} such that $(q_m, q_{m+1}) \in E$. A *complete play* may be finite or infinite, but if it is finite and ends in some vertex q_m, we require this vertex to be *terminal* in the sense that there is no vertex r such that $(q_m, r) \in E$. If the play ρ is infinite, we define the set of vertices occurring infinitely often by

$$\mathrm{Inf}(\rho) \stackrel{\text{def}}{=} \{q \in Q \mid \exists^{\omega} i \rho(i) = q\}$$

Graph games can be equipped with a variety of winning conditions, but in this dissertation we consider three: *finite*, *Muller* and *parity* conditions.

A *finite graph game* (G, F) is equipped with a subset $F \subseteq Q$ of accepting vertices. We assume that there are no *cycles* in G; under this assumption, every play $\rho = q_0 q_1 \ldots$ of (G, F) has $q_i \neq q_j$ for $j < i$ (i.e. no vertex is visited twice). Clearly this entails that every complete play of (G, F) is finite as the set of vertices is finite. Player I wins the complete play ρ if it ends in some vertex $q_m \in F$ and Player II wins otherwise.

A *Muller game* $(G, \mathcal{F})$ is equipped with a collection $\mathcal{F} \subseteq \mathcal{P}(Q)$ of subsets of the vertices Q. In a Muller game, we assume that every vertex q has an outgoing edge, i.e. there exists some r such that $(q, r) \in E$. This assumption means that there are no terminal vertices (unlike finite graph games) and hence all complete plays of Muller games are infinite. Player I wins the complete play ρ if $\mathrm{Inf}(\rho) \in \mathcal{F}$ and Player II wins otherwise.

A *parity game* (G, c) is equipped with a colouring function $c : Q \to \{0, \ldots, k\}$ for some finite k. As in Muller games, we assume that there are no terminal vertices and hence all

complete plays are infinite. Player I wins the complete play ρ if $\max\{c(q) \mid q \in \mathrm{Inf}(\rho)\}$ (the maximal colour appearing infinitely often in ρ) is even and Player II wins otherwise.

In each case, we say that a strategy s is *winning* for Player I from vertex q (respectively Player II) if every complete play from q where the moves following Player-I (respectively Player-II) vertices are generated according to s is won by Player I (respectively Player II). We say that q is in the *winning region* for Player I (respectively Player II) if there exists a winning strategy for Player I (respectively Player II) from q. We say that a graph game is *determined* if every vertex belongs to the winning region of one of the two players.

For a set of vertices V, we wish to define the set of vertices from which Player I can be sure to reach some vertex in V regardless of the actions of Player II. We build this set using a backwards reachability or *attractor* computation as follows:

$$\begin{aligned} Attr_I^0(V) &= V \\ Attr_I^{i+1}(V) &= Attr_I^i(V) \\ &\quad \cup \{q \in Q_I \mid \exists (q,r) \in E, r \in Attr_I^i(V)\} \\ &\quad \cup \{q \in Q_{II} \mid \forall (q,r) \in E, r \in Attr_I^i(V)\} \end{aligned}$$

Then Player I's attractor for V is defined by $Attr_I(V) = \bigcup_{i=0}^{|Q|} Attr_I^i(V)$ as no more vertices can be added after $|Q|$ steps. We similarly define Player II's attractor for V by reversing the roles of the players in this construction. We say that Player I uses an *attractor strategy* for V if at every vertex in $Attr_I^i(V)$ for $i > 0$ he chooses moves which ensure that the next vertex is in $Attr_I^{i-1}(V)$. Note that such an attractor strategy can be *memoryless*, that is, the move chosen at each vertex only needs to depend on that vertex and not the previous vertices in the partial play.

Proposition 2.15. *A parity game (G, c) is determined, and one can compute the winning regions W_I and W_{II} and also compute corresponding memoryless winning strategies of Player I and Player II.*

Proof Sketch. For this proposition, we follow a proof of McNaughton [McN93]. The proof proceeds by induction on the size of Q, the set of vertices of G. Assume that the maximal colour k is even (otherwise reverse the roles of Player I and Player II in the following). Let q be a vertex of maximal colour and let $A = Attr_I(\{q\})$. The set of vertices $Q \setminus A$ induces a subgame and by the induction hypothesis we let the winning regions of this subgame be U_I and U_{II}. In one step from q, it is easy to see that either Player I can be sure to be in $U_I \cup A$ or Player II can be sure to be in U_{II}.

In the first case, we note that $U_I \cup A \subseteq W_I$ since in every play Player I can ensure the game remains forever in U_I or otherwise periodically visits q by using the attractor strategy for q. In this case, $W_{II} = U_{II}$.

In the second case, we consider the set $B = Attr_{II}(U_{II} \cup \{q\})$ and the decomposition of the subgame induced by $Q \setminus B$ into winning regions V_I and V_{II}. Then $W_I = V_I$ and $W_{II} = V_{II} \cup B$. □

Proposition 2.16. *A Muller game $(G, \mathcal{F})$ is determined, and one can compute the winning regions W_I and W_{II} and also compute corresponding finite-state winning strategies of Player I and Player II.*

Note that while Büchi and Landweber provided a direct proof of this proposition in [BL69], Muller games can be reduced to parity games by employing a structure known as a "latest appearance record", which serves to record in the vertices of the game the finite-state memory needed by the two players [McN65, GH82].

2.7.3 Solving McNaughton Games

Having defined the auxiliary formalism of graph games, we can now solve the synthesis problem for McNaughton games.

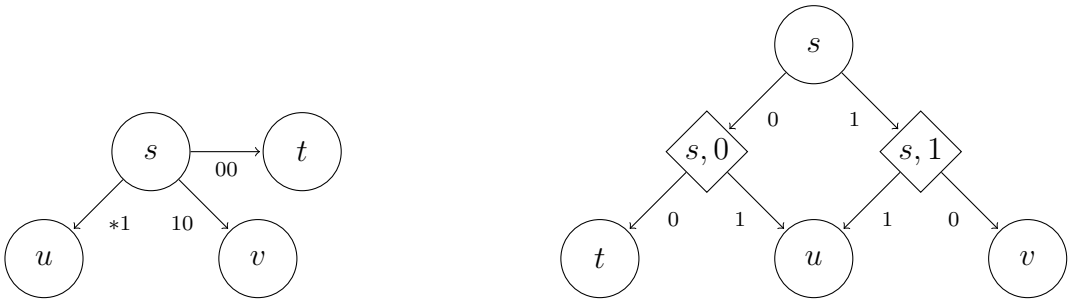
Theorem 2.17. *A McNaughton game $\mathbb{G}(\varphi)$ is determined and one can compute a finite-state winning strategy for the winning player.*

Proof. In order to solve the synthesis problem, we first translate the formula φ into a finite automaton $\mathcal{A}_\varphi$; this translation proceeds by induction on the structure of φ [Büc62]. The input to $\mathcal{A}_\varphi$ represents a play of the game; each letter is a pair of the contributions of the two players in a particular round (and is therefore from the alphabet $\{0,1\}^2$). This automaton will in general be nondeterministic because of the translation of existential quantification and disjunction.

The next step is to determinise this automaton, so we must consider $\mathcal{A}_\varphi$ to have a Muller acceptance condition (the simpler Büchi automata cannot be determinised in general but Muller automata can; see the survey [Far02] for details). Muller acceptance requires that the set $\mathrm{Inf}(\rho)$ of states which occur infinitely often in a run ρ is exactly one of a family of accepting sets $\mathcal{F}$. Note that a Büchi automaton $(S, s_0, \Sigma, F, \Delta)$ can be considered as a Muller automaton with F replaced by the accepting family $\{S' \subseteq S \mid S' \cap F \neq \emptyset\}$.

Now considering $\mathcal{A}_\varphi$ as a deterministic Muller automaton, we can construct a Muller game M_φ which intuitively separates the contribution of the two players of $\mathbb{G}(\varphi)$. The vertices of this game are $Q = S \cup S \times \{0,1\}$ with the vertices $Q_I = S$ under the control of Player I. For a transition $\Delta(s,(b_1,b_2)) = t$ in the automaton $\mathcal{A}_\varphi$, we define two edges $(s,(s,b_1)) \in E$ and $((s,b_1),t) \in E$ in the Muller game which represent Player I first choosing a bit of the input and Player II completing the transition by choosing a bit in response. We retain the accepting family $\mathcal{F}$ as the winning condition of the Muller game, so we have $M_\varphi = ((Q, Q_I, E), \mathcal{F})$. An example of this process can be seen in Figure 2.5 where the Player-I vertices are represented as circles and the Player-II vertices as diamonds.

It is straightforward that following this construction Player I wins the McNaughton game $\mathbb{G}(\varphi)$ if and only if he wins the Muller game M_φ from s_0 (the initial state of the intermediate Muller automaton) and since his contribution is the same at each stage of



(a) Fragment of Automaton $\mathcal{A}_\varphi$

(b) Fragment of Muller Game M_φ

Figure 2.5: Transforming a Muller automaton into a game graph

both games, a strategy for M_φ can easily be transformed into one which applies to $\mathbb{G}(\varphi)$ (and similarly for Player II). Hence, using the construction of Proposition 2.16 to obtain a finite-state winning strategy for the player who wins M_φ from s_0, we obtain a finite-state strategy to win $\mathbb{G}(\varphi)$. □

Chapter 3

Time-Bounded Alternating Timed Automata

3.1 Introduction

Even over finite words, the language emptiness problem for alternating timed automata with multiple clocks is undecidable [ADOW05] as these automata can simulate any finite computation of a Turing machine. If the automata are restricted to having only a single clock, decidability of the finite word language emptiness problem is restored [OW05, LW05], but these machines cannot model multiple independent timing constraints on a system in an intuitive way. Having multiple clocks allows this modelling power, as can be seen in the example of Section 2.4.3, and we explore in this chapter a restriction of alternating timed automata which retains multiple clocks and whose language emptiness problem is decidable.

In [ORW09], Ouaknine, Rabinovich and Worrell considered the idea of *time-bounded* verification in which all timestamps associated with events of the system come from a fixed, bounded interval of the reals. The authors proved that this restriction restores the

decidability of the *language inclusion* problem for nondeterministic timed automata as well as the satisfiability problem for monadic second-order logic over finitely variable predicates.

The notion of time-bounded verification can be seen as analogous to the classical idea of *bounded model checking* which attempts to restore tractability to the problem of ensuring that a system meets a specification by considering only a bounded number of events affecting the system. Note, however, that time-bounded verification differs from this approach in as much as it considers an arbitrary number of events due to the density of time — only the maximum value of the timestamps of the events is bounded.

This restriction to time-bounded executions of a system can be seen as harmless in many contexts, as real-world systems are expected to run for only a finite duration before being shut down or restarted. For example, a run of a communication protocol may have an *a priori* timeout, or an avionics system may be started fresh before each flight, but in both cases the number of events which must be considered during an execution of the system is potentially unbounded.

The approach to solving the language inclusion problem in [ORW09] is to compute a formula $\varphi_{\mathcal{A}}$ which defines the same language as the timed automaton $\mathcal{A}$, thus following the method of Henzinger, Raskin and Schobbens [HRS98] and reducing the automata problem to one of logical satisfiability. Intuitively, the formula $\varphi_{\mathcal{A}}$ can be defined by using additional predicates to represent a run of $\mathcal{A}$ on the timed word. These predicates are existentially quantified out, as the existence of an accepting run of the timed automaton is equivalent to the existence of a satisfying assignment to these predicates.

This approach, therefore, does not cleanly extend to alternating timed automata whose semantics are presented as a game; a word is accepted by an alternating timed automaton if one of the players (Automaton) has a winning strategy in this game. We can simulate a single play of the game with a formula in the same way, but since acceptance of a word corresponds to a winning strategy rather than a single winning play of the game, we would

need to represent the whole game tree. We therefore move instead to a game-theoretic approach and present in Section 3.5.1 a formula which is then used as the winning condition in a *McNaughton game*.

A McNaughton game [McN65] builds assignments of predicates by alternating moves of two players, one of whom wishes the assignment to satisfy a winning condition and one of whom wishes the opposite. McNaughton used this type of game over the natural numbers in work on a solution to Church's *synthesis problem* as explained in Section 2.7.

Modelling the acceptance game of an alternating timed automaton on a time-bounded word is simpler than a McNaughton game in one important respect: the word must be *finite* because we require it to be *non-Zeno*, i.e., only finitely many timed events can occur in the bounded interval of time despite their number being potentially unbounded. We identify, however, three ways in which McNaughton games must be extended in order to model the acceptance game situation.

Firstly, each word input to the automaton affects the acceptance game. Since we wish to consider the language emptiness problem, we define a family of games *parameterised* by the input word. McNaughton games with parameters over the naturals were considered by Rabinovich [Rab07a] and later by Hänsch, Slaats and Thomas [HST09], and we consider here an extension to finite games over the reals. We determine in Section 3.4.2 the set of parameters for which one player can be sure to win the corresponding games.

The handling of real-time is our second extension, but since we consider these McNaughton games only to be a technical tool, we adopt the simplifying convention that the players move exactly when one of the parameters becomes true. This simplification is required to avoid Zeno plays which might be generated if either of the players were allowed to choose the time points at which moves are made; this problem was considered by de Alfaro et al. for games without parameters [dAFH+03].

Finally, we need to handle the metric nature of real-time. While it is possible to remove

the +1 relation from the logic if only a bounded interval of the reals is considered [ORW09], this process destroys the causality captured by the game structure. Our solution is presented in Section 3.5.3 and considers a series of games, one for each time unit, which collectively construct the predicates required. Using this construction, we conclude that the time-bounded language emptiness problem is decidable for alternating timed automata.

We note, however, that this construction introduces an exponential blow-up per time unit and show in Section 3.6 that it is possible to simulate a 2-counter machine with counters whose size is bounded by a tower of exponentials — the height of the tower equals the number of time units. Thus the time-bounded language emptiness problem for alternating timed automata has non-elementary complexity in the time bound, which stands in contrast to the situation for nondeterministic timed automata where time-bounded language inclusion (which subsumes both language emptiness and the dual universality problem) is 2-ExpSpace-complete [ORW09].

3.2 Alternating Timed Automata

In this chapter, we concern ourselves with timed words that are *bounded*, which is to say that given a time bound $N \in \mathbb{R}_+$, we consider words $w = (\sigma, \tau)$ such that $\tau_i < N$ for all timestamps τ_i in τ. As we observe in Section 2.2, combining this restriction with the natural requirement that timed words be *non-Zeno* means that we consider only finite bounded timed words.

Recall from Section 2.4 that an alternating timed automaton $\mathcal{A}$ has a set S of locations, with a unique initial location s_0 and a subset F of accepting locations. It augments these with a set of clock variables C which can assume any positive real value. The automaton consumes timed events, which consist of letters from the alphabet Σ augmented with timestamps in $\mathbb{R}_+$, and its transition function $\delta : S \times \Sigma \times \Phi(C) \to \mathcal{B}^+(\mathit{Moves}(C, S))$ maps each

combination of location, letter and clock constraint to a positive Boolean combination of moves, which we assume is presented in disjunctive normal form. $Moves(C, S) = \mathcal{P}(C) \times S$, so each such move consists of a next location together with a subset of clocks to be reset when this move is made.

The semantics of such an automaton is defined in terms of a game $G(\mathcal{A}, w)$ between *Automaton* and *Pathfinder* on finite input $w = (\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)$. The positions of the game are triples (s, ν, i) consisting of the current automaton location $s \in S$ and clock valuation $\nu : C \to \mathbb{R}_+$ along with the round number $i \in \mathbb{N}$. There is one round corresponding to each timed event in w. For a round $i + 1$ starting in position (s_i, ν_i, i), we assume that the automaton is structured such that there is at most one transition $\delta(s_i, \sigma_{i+1}, \varphi_{i+1})$ available this round (the *Disjointness* assumption). By this we mean that when we update the clocks with the delay to the event $(\sigma_{i+1}, \tau_{i+1})$ seen this round, the constraint φ_{i+1} is met but there is no other transition $\delta(s_i, \sigma_{i+1}, \lambda)$ where the constraint λ is met, i.e. $\nu_i + (\tau_{i+1} - \tau_i) \in [\varphi_{i+1}]$ but $\nu_i + (\tau_{i+1} - \tau_i) \notin [\lambda]$. Automaton then chooses a conjunction from $\delta(s_i, \sigma_{i+1}, \varphi_{i+1})$ and Pathfinder chooses a move from that conjunction, say (Y, s_{i+1}). We update the game accordingly and start the next round in position $(s_{i+1}, \nu_{i+1}, i + 1)$, where $\nu_{i+1} = [Y := 0](\nu_i + (\tau_{i+1} - \tau_i))$. If Automaton is ever unable to choose a conjunction (because no transition is defined), he loses. Otherwise, the game ends in some position (s_n, ν_n, n) and Automaton wins if $s_n \in F$.

We then say that $\mathcal{A}$ accepts w if Automaton has a winning strategy in $G(\mathcal{A}, w)$ and that $L_f(\mathcal{A})$ is the set of finite timed words accepted by $\mathcal{A}$. We will usually omit the subscript and refer to $L(\mathcal{A})$ as throughout this chapter we deal only with finite timed words. We now introduce the key problem considered in this chapter.

Definition 3.1 (Time-Bounded Language Emptiness Problem)**.** Given a time bound N and an alternating timed automaton $\mathcal{A}$, decide whether there exists any timed word $w = (\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)$ with $\tau_i \in [0, N)$ for all $1 \leq i \leq n$ such that $w \in L(\mathcal{A})$.

3.3 Weak Monadic Second-Order Logic

Recall from Section 2.6 that we consider $\mathsf{MSO}(<, +1)$ over a signature consisting of the binary relations $<$ and $+1$ and a countable family of monadic predicate names $P_0, P_1, \ldots$. We denote by $\mathsf{MSO}(<)$ the sub-language consisting of all formulae that do not mention the $+1$ relation. We consider structures of the form $\mathcal{M} = \langle \mathbb{T}, <, +1, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$, where $\mathbb{T}$ is an interval of non-negative reals with the usual order and $\mathbf{P}_1, \ldots, \mathbf{P}_m$ are subsets of $\mathbb{T}$ that interpret the monadic predicate names $P_1, \ldots, P_m$. Recall that we use boldface to denote the interpretations of predicate names.

Shelah [She75] famously proved that the satisfiability of $\mathsf{MSO}(<)$ is undecidable over $\langle \mathbb{R}_+, < \rangle$. It is not hard to see that the result also holds for every bounded interval in $\mathbb{R}_+$ since such an interval contains a definable subset that is order-isomorphic to $\mathbb{R}_+$. For example, the interval $[0, 1]$ contains the definable subset $[0, 1)$ which is order-isomorphic to $\mathbb{R}_+$. Shelah also proved that decidability can be restored if we restrict consideration to countable predicates, but since much of this chapter deals with bounded time, we consider the stronger condition of dealing only with finite predicates whenever decidability of the logic is to be used.

Definition 3.2. Weak MSO (WMSO) shares the same syntax as MSO, but all free predicate variables are interpreted only as finite sets and second-order quantification ranges only over finite sets.

Decidability of $\mathsf{MSO}(<)$ under this restriction follows from Shelah's result mentioned above; cf. also [Rab02]. Note that we do not restrict the domain over which the logic is interpreted so this need not be finite.

Let $\Sigma_P = \{0, 1\}^m$ be a finite alphabet and $\mathbb{T}$ be an interval of non-negative reals. A structure $\langle \mathbb{T}, <, +1, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ for Weak MSO corresponds to a timed word over Σ_P in the following sense. Let $\tau = (\tau_i)_i$ be the sequence of points where some $\mathbf{P}_j$ holds, sorted in

ascending order. Further, let $\sigma_i = (\mathbf{P}_1(\tau_i), \ldots, \mathbf{P}_m(\tau_i)) \in \Sigma_P$ then let $\sigma = (\sigma_i)_i$. Then we say that (σ, τ) is the timed word corresponding to this structure and for any $\mathsf{MSO}(<, +1)$ sentence $\varphi(\overline{P})$, $(\sigma, \tau) \models \varphi$ if and only if $\langle \mathbb{T}, <, +1, \overline{\mathbf{P}} \rangle \models \varphi$. Note that (σ, τ) is a finite timed word since each $\mathbf{P}_j$ is a finite subset of $\mathbb{T}$.

Moreover, observe that every finite timed word $w = (\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n)$ over alphabet Σ_P defines a set of interpretations $\overline{\mathbf{P}}$ by $\mathbf{P}_i = \{\tau_j \mid \sigma_j(i) = 1\}$. Hence we associate w with the structure $\mathcal{M} = \langle \mathbb{T}, <, +1, \overline{\mathbf{P}} \rangle$.

We denote by $\mathbb{T}\Sigma^*$ the set of all finite timed words over an alphabet Σ with timestamps in $\mathbb{T}$. A $\mathsf{WMSO}(<, +1)$ sentence $\varphi(\overline{P})$ that mentions predicate names $P_1, \ldots, P_m$ defines a language of timed words $L(\varphi) \stackrel{\text{def}}{=} \{w \in \mathbb{T}\Sigma_P^* \mid w \models \varphi\}$ for alphabet $\Sigma_P = \{0, 1\}^m$.

If we furthermore assume, without loss of generality, that $\overline{P}$ contains a distinguished predicate P_{init} with a fixed interpretation as the set $\{0\}$, then for timed words w and w', the structures $\mathcal{M}_w$ and $\mathcal{M}_{w'}$ are order isomorphic if and only if $\mathit{Untime}(w) = \mathit{Untime}(w')$.[1] Building on this observation, we can represent the set of models of an $\mathsf{WMSO}(<)$ formula (up to order isomorphism) as a regular untimed language.

Definition 3.3. A collection $\mathcal{C}$ of interpretations of $P_1, \ldots, P_m$ in A is said to be *regular* if there exists a regular (untimed) language L on alphabet $\Sigma_P = \{0, 1\}^m$ such that

$$\mathcal{C} = \{\mathcal{M}_w \mid w \in \mathbb{T}\Sigma_P^*,\ \mathit{Untime}(w) \in L\}\,.$$

We say that L corresponds to $\mathcal{C}$.

The following proposition follows from [Rab02, Theorem 1].

Proposition 3.4. *1. For every formula $\varphi(\overline{P}) \in \mathsf{WMSO}(<)$ one can compute a finite state automaton $\mathcal{A}_\varphi$ such that $L(\mathcal{A}_\varphi)$ corresponds to the set of models of φ.*

2. For every regular language L one can compute a formula $\varphi(\overline{P}) \in \mathsf{WMSO}(<)$ such that L corresponds to the set of models of φ.

[1] We include the predicate P_{init} since an order isomorphism from $\mathcal{M}_w$ to $\mathcal{M}_{w'}$ must map 0 to itself.

3.4 Real-Time McNaughton Games

Observing the similarity between the McNaughton game defined in Section 2.7.1 and the acceptance game for alternating timed automata defined in Section 2.4.2, we wish to define a bounded real-time extension of McNaughton games in which to encode the acceptance game. Since the acceptance game includes a word, it is natural to consider this as a set of *parameters* to the McNaughton game. The word will determine the number of rounds of the game and, corresponding to our exclusion of Zeno time sequences, we wish a play of the game to have only a finite number of rounds.

3.4.1 Defining a Game

For the rest of this section, we consider a fixed structure $\mathcal{M} = \langle [0, N), <, +1, \overline{\mathbf{P}} \rangle$, where $N \in \mathbb{N}$, $[0, N) \subset \mathbb{R}_+$, $<$ is the usual binary order relation, $+1(x, y)$ holds iff $x + 1 = y$ and $\overline{\mathbf{P}} = \mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_p$ is a vector of finite sets. We intend these $\mathbf{P}_i$ to be the interpretation of specific predicate variables (the *parameters*) in our formula.

Recall from Section 3.3 that Weak MSO interprets all predicate variables as finite sets and restricts quantification to range only over finite sets; this restriction is ideally suited here. Since each predicate variable represents only a finite set, we need only a finite number of rounds to define the time points in the set (and we assume all other time points are excluded). The only question to resolve is who gets to choose the time point of each round. We present here a simple resolution - the players move only at time points where some parameter holds. Note that this is sufficient for encoding the acceptance game as the parameters will represent the letters input to the alternating timed automaton.

Let $\varphi(\overline{P}, \overline{X}, \overline{Y})$ be a $\mathsf{WMSO}(<, +1)$ formula where $\overline{P} = P_1, P_2, \ldots, P_p$, $\overline{X} = X_1, X_2, \ldots, X_x$ and $\overline{Y} = Y_1, Y_2, \ldots, Y_y$ are vectors of monadic predicate variables. Let $\mathbf{T} = \bigcup_{i=1}^{p} \mathbf{P}_i$; this is the finite set of points where some interpretation of a parameter variable holds. Write

$\mathbf{T} = \{t_0, \ldots, t_{n-1}\}$ with $t_i < t_{i+1}$ for each i.

The *real-time McNaughton game* $\mathbb{G}(\mathcal{M}, \varphi)$ is defined as follows:

1. The game is played by two players, called Player I and Player II.
2. There is one round for each $t_i \in \mathbf{T}$.
3. In the ith round, Player I chooses a value $\overline{\mathbf{X}}(t_i) \in \{0,1\}^x$ then Player II chooses $\overline{\mathbf{Y}}(t_i) \in \{0,1\}^y$.
4. At the conclusion of the game, we set $\overline{\mathbf{X}}(t) = \overline{0}$ and $\overline{\mathbf{Y}}(t) = \overline{0}$ for $t \notin \mathbf{T}$ to complete the construction of the two vectors of finite predicates $\overline{\mathbf{X}} \subseteq [0,N)^x$ and $\overline{\mathbf{Y}} \subseteq [0,N)^y$.
5. Player I wins the play if $\langle \mathcal{M}, \overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}} \rangle \models \varphi$, otherwise Player II wins the play.

In order to establish the decidability of the language emptiness problem for alternating timed automata, we consider first the case that $\varphi \in \mathsf{WMSO}(<)$, i.e. it is purely order-theoretic. In this case we wish to compute the *sets of parameters* for which a McNaughton game can be won by Player I. We show that the following is computable:

Definition 3.5 (Parameter Problem)**.** Given $\langle [0,N), < \rangle$ and $\varphi(\overline{P}, \overline{X}, \overline{Y})$, compute a formula $\theta(\overline{P})$ such that for each interpretation $\overline{\mathbf{P}}$ of $\overline{P}$ and corresponding structure $\mathcal{M} = \langle [0,N), <, \overline{\mathbf{P}} \rangle$, $\mathcal{M} \models \theta$ if and only if Player I has a winning strategy in $\mathbb{G}(\mathcal{M}, \varphi)$.

We use this result to answer the question of whether there exists any interpretation of the parameters for which Player I has a winning strategy in the case where $\varphi \in \mathsf{WMSO}(<, +1)$ and reduce the language emptiness problem for alternating timed automata to this problem.

3.4.2 Solving the Parameter Problem

Given $\varphi(\overline{P}, \overline{X}, \overline{Y}) \in \mathsf{WMSO}(<)$ and an interval $[0,N)$, we wish to compute the set of values of $\overline{\mathbf{P}}$ for which Player I has a winning strategy in the McNaughton game $\mathbb{G}(\mathcal{M}, \varphi)$, where $\mathcal{M} = \langle [0,N), <, \overline{\mathbf{P}} \rangle$.

Recall from Definition 3.3 that, thanks to the interpretation of predicates as finite sets, we can represent the set of interpretations satisfying an $\mathsf{WMSO}(<)$ formula $\varphi(\overline{X})$ as a regular untimed language.

Proposition 3.6. *Given a* $\mathsf{WMSO}(<)$ *formula* $\varphi(\overline{P}, \overline{X}, \overline{Y})$*, one can compute a formula* $\theta(\overline{P})$ *such that for* $\mathcal{M} = \langle [0, N), <, \overline{\mathbf{P}} \rangle$*, Player I wins* $\mathbb{G}(\mathcal{M}, \varphi)$ *if and only if* $\mathcal{M} \models \theta$*.*

Proof. According to Proposition 3.4 (1) we can compute a deterministic finite automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ over alphabet $\Sigma = \{0,1\}^{\overline{P}} \times \{0,1\}^{\overline{X}} \times \{0,1\}^{\overline{Y}}$ whose language represents the set of models of φ. We seek an automaton over alphabet $\Sigma_P = \{0,1\}^{\overline{P}}$ representing the set of interpretations $\overline{\mathbf{P}}$ of $\overline{P}$ such that Player I wins $\mathbb{G}(\mathcal{M}, \varphi)$ for $\mathcal{M} = \langle [0, N), <, \overline{\mathbf{P}} \rangle$.

Let $w = w_1 w_2 \dots w_n \in \Sigma_P^*$ represent an interpretation of $\mathbf{P}$. By distinguishing the contribution of input bits between two protagonists, respectively called Player I and Player II, we define a *finite graph game* $\Gamma(\mathcal{A}, w) = ((V, V_I, E), W)$, which can be seen as a discrete analogue of $\mathbb{G}(\mathcal{M}, \varphi)$ with $\mathcal{M} = \langle [0, N), <, \overline{\mathbf{P}} \rangle$.

For each automaton state $q \in Q$ and position $0 \leq i \leq n$, we include a Player-I vertex $(q, i) \in V_I$; if moreover $0 \leq i \leq n-1$ then we also include a Player-II vertex $(q, i, b) \in (V \backslash V_I)$ for each bit vector $b \in \{0,1\}^{\overline{X}}$. If $0 \leq i \leq n-1$ then we include an edge in E from (q, i) to (q, i, b), corresponding to a choice of bit vector b by Player I; for $b' \in \{0,1\}^{\overline{Y}}$ we also include an edge in E from (q, i, b) to $(q', i+1)$, where $q' = \delta(q, (w_{i+1}, b, b'))$, corresponding to a choice of bit vector b' by Player II. We let the winning set $W \stackrel{\text{def}}{=} \{(q, n) \mid q \in F\}$.

As in Section 2.7.2, the rules of the game $\Gamma(\mathcal{A}, w)$ are as follows: Player I chooses moves at vertices in V_I and Player II chooses moves at vertices in $V \setminus V_I$. Play starts in the vertex $(q_0, 0)$ and Player I wins if play reaches a vertex in W.

The game $\Gamma(\mathcal{A}, w)$ essentially represents an *untiming* of $\mathbb{G}(\mathcal{M}, \varphi)$: the moves in each game are the same once one elides the timestamp associated with each round in the latter game. In particular, Player I wins $\Gamma(\mathcal{A}, w)$ if and only if Player I wins $\mathbb{G}(\mathcal{M}, \varphi)$. We now consider the word w as a parameter to this game and seek a solution to the whole family

of finite graph games $\Gamma(\mathcal{A}, w)$ for different values of w.

For $E \subseteq Q$ a set of automaton states and parameter word $w \in \Sigma_P^*$, we define the set $\text{Force}_w(E)$ of states from which Player I can force play into E on input w as follows:

$$\begin{aligned}
\text{Force}_\varepsilon(E) &= E \\
\text{Force}_u(E) &= \{q : \exists b_1 \, \forall b_2 \, \delta(q, (u, b_1, b_2)) \in E\} \quad u \in \Sigma_P \\
\text{Force}_{uw}(E) &= \text{Force}_u(\text{Force}_w(E)) \qquad u \in \Sigma_P, w \in \Sigma_P^*
\end{aligned}$$

Given $w \in \Sigma_P^*$, it is straightforward that $q_0 \in \text{Force}_w(F)$ if and only if Player I wins $\Gamma(\mathcal{A}, w)$. From this observation one can build an automaton $\mathcal{B}$ on alphabet Σ_P that accepts those words w such that Player I wins $\Gamma(\mathcal{A}, w)$. The set of states of $\mathcal{B}$ is $\mathcal{P}(Q)$, with F the unique final state, and $\{S \subseteq Q \mid q_0 \in S\}$ the set of initial states. We include a transition $S \xrightarrow{b} T$ on input $b \in \Sigma_P$ if and only if $S = \text{Force}_b(T)$.

By Proposition 3.4 (2), the equivalence of $\mathsf{WMSO}(<)$ and automata, we can compute a formula $\theta(\overline{P})$ such that the set of models of θ corresponds to $L(\mathcal{B})$. □

A result similar to Proposition 3.6 has been proven in [HST09, Rab07a] for parametric games over $(\mathbb{N}, <)$.

3.5 Solving the Acceptance Game

In this section, we reduce the acceptance game $G(\mathcal{A}, w)$ for an alternating timed automata $\mathcal{A} = (S, s_0, \Sigma, C, F, \delta)$ and a finite timed word $w = (\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)$ into a sequence of bounded real-time McNaughton games as introduced in Section 3.4.1. The general idea is to transform the automaton into the winning condition for the play, the choices of Automaton and Pathfinder to those of Players I and II respectively and the word w into a set of parameters.

Let $\tau_n < N \in \mathbb{N}$. It is clear that the acceptance game will terminate before N, so we expect the McNaughton game to do likewise and consider it over the interval $[0, N)$; for the rest of this section we fix the structure $\mathcal{M} = \langle [0, N), <, +1 \rangle$.

3.5.1 From Automata to Logic

We first recall from Section 3.3 that each finite timed word w can be associated with a finite set of interpretations $\overline{\mathbf{P}}$ and a structure $\mathcal{M}$.

In order to encode the choices of Automaton in the acceptance game, we write A for the subset of $\mathit{Moves}(C, S)$ occurring in the alternating timed automaton $\mathcal{A}$ and enumerate $\mathcal{P}(A)$ as $\{X_i \mid 1 \leq i \leq x\}$ for some $x \in \mathbb{N}$. These are the possible models that Automaton may choose during $G(\mathcal{A}, w)$, so each choice in this game corresponds to one such X_i. The X_i can therefore be considered as predicates which are true precisely when the corresponding model is chosen by Automaton. The choices of Pathfinder are simply elements of $\mathit{Moves}(C, S)$, so they can similarly be captured by enumerating A as $\{Y_i \mid i \in 1 \leq i \leq y\}$ for some $y \in \mathbb{N}$. Since any acceptance game only has a finite number of rounds, all of these predicates represent finite sets of time points where a move is played.

We define a formula $\varphi_{\mathcal{A}}(\overline{P}, \overline{X}, \overline{Y})$ and imagine that Player I and Player II in $\mathbb{G}(\mathcal{M}_w, \varphi_{\mathcal{A}})$ respectively take the roles of Automaton and Pathfinder in $G(\mathcal{A}, w)$. We instrument the formula $\varphi_{\mathcal{A}}$ so that Player I wins the McNaughton game $\mathbb{G}(\mathcal{M}_w, \varphi_{\mathcal{A}})$ if and only if Automaton wins the acceptance game $G(\mathcal{A}, w)$. The key components of $\varphi_{\mathcal{A}}$ are subformulae $\varphi_{\text{aut}}(t)$ and $\varphi_{\text{path}}(t)$, respectively ensuring that Player I and Player II correctly simulate Automaton and Pathfinder at each time point t.

The formula $\varphi_{\text{path}}(t)$ ensures that at time t Pathfinder only chooses one move α, and moreover that α is a conjunct of the model M chosen by Automaton at time t (written

$\alpha \in M$). This is expressed as:

$$\bigwedge_{\alpha} \left(Y_\alpha(t) \rightarrow \bigvee_{\alpha \in M} X_M(t) \right) \wedge \bigwedge_{\alpha \neq \beta} \neg(Y_\alpha(t) \wedge Y_\beta(t)) \,.$$

The formula $\varphi_{\text{aut}}(t)$ ensures that at time t Automaton chooses a model M of the transition function δ. It is the conjunction over all locations $s \in S$, inputs $\mu_i \in \Sigma$ and guards $\psi \in \Phi(C)$ such that $\delta(s, \mu_i, \psi)$ is defined, of the formulae

$$\forall v \left((\mathit{state}_s(v) \wedge \mathit{next}(v,t) \wedge P_i(t) \wedge \mathit{const}_\psi(t)) \rightarrow \bigvee_{M \models \delta(s,\mu_i,\psi)} X_M(t) \right) .$$

Here $\mathit{state}_s(v)$ and $\mathit{next}(v,t)$ are easily defined auxiliary formulae, respectively expressing that the automaton is in state s at time v, and that v and t are consecutive timestamps in the input word. Similarly, $\mathit{const}_\psi(t)$ expresses that the clock constraint $\psi \in \Phi(C)$ holds at time t. For example, in the case where $\psi \equiv x \sim k$ we define $\mathit{const}_\psi(t)$ to be the formula

$$\exists u \left(\begin{array}{c} u < t \wedge \mathit{reset}_x(u) \wedge \forall w(u < w < t \rightarrow \neg \mathit{reset}_x(w)) \\ \wedge\, t - u \sim k \end{array} \right)$$

where $\mathit{reset}_x(u)$ is an auxiliary formula expressing that clock x was reset at time u (which information is available from $Y_\alpha(u)$).

φ_{aut} and φ_{path} are components of a formula $\varphi_{\mathcal{A}}$ which encodes the winning condition of Automaton in the acceptance game. Specifically, $\varphi_{\mathcal{A}}$ expresses that play must start in an initial location, it must end in an accepting location, and for all time points t at which some P_i holds (i.e., all time points of the input word) $\varphi_{\text{aut}}(t)$ holds unless φ_{path} had previously failed. This is all straightforward to formalise, and in fact this can be accomplished such that $\varphi_{\mathcal{A}}$ has no second-order quantifiers, that is, it is a formula in the first-order fragment of $\mathsf{MSO}(<, +1)$.

The games $G(\mathcal{A}, w)$ and $\mathbb{G}(\mathcal{M}_w, \varphi_{\mathcal{A}})$ are essentially isomorphic, and it is now straightforward to prove the following:

Proposition 3.7. *Automaton wins the acceptance game $G(\mathcal{A}, w)$ if and only if Player I wins the McNaughton game $\mathbb{G}(\mathcal{M}_w, \varphi_{\mathcal{A}})$, where $\mathcal{M}_w$ is the structure associated with the timed word w.*

Next, we remove the $+1$ relation by using the stacking construction presented in [ORW09] to transform the formula.

3.5.2 Eliminating the Metric

Given an $\mathsf{MSO}(<, +1)$ formula φ, we define a straightforward syntactic transformation into an $\mathsf{MSO}(<)$ formula $\overline{\varphi}$ such that there is a natural bijection between models of φ with domain $[0, N)$ and models of $\overline{\varphi}$ with domain $[0, 1)$.

With each monadic predicate X that appears in φ, we associate a collection $X_0, \ldots, X_{N-1}$ of N fresh monadic predicates. Intuitively, each X_i is a predicate on $[0, 1)$ that represents X over the subinterval $[i, i+1)$. Formally, an interpretation of X over domain $[0, N)$ yields interpretations of the X_i over $[0, 1)$ by defining $X_i(t)$ if and only if $X(i+t)$. Note that this correspondence yields a bijection between interpretations of X on $[0, N)$ and interpretations of $X_0, \ldots, X_{N-1}$ on $[0, 1)$.

We can assume that φ does not contain any (first- or second-order) existential quantifiers by replacing them with combinations of universal quantifiers and negations if need be. It is also convenient to rewrite φ into a formula that makes use of a unary function $+1$ instead of the $+1$ relation. To this end, replace every occurrence of $+1(x, y)$ in φ by $(x < N - 1 \wedge x + 1 = y)$.

Next, replace every instance of $\forall x\, \psi$ in φ by the formula

$$\forall x \left(\psi[x/x] \wedge \psi[x+1/x] \wedge \ldots \wedge \psi[x+(N-1)/x]\right),$$

where $\psi[t/x]$ denotes the formula resulting from substituting every free occurrence of the variable x in ψ by the term t. Intuitively, this transformation is legitimate since first-order

variables in our target formula will range over $[0, 1)$ rather than $[0, N)$.

Having carried out these substitutions, use simple arithmetic to rewrite every term in φ as $x + k$, where x is a variable and $k \in \mathbb{N}$ is a non-negative integer constant.

Every inequality occurring in φ is now of the form $x + k < N - 1$ or $x + k_1 < y + k_2$. Replace every inequality of the first kind by **true** if $k + 2 \leq N$ and by **false** otherwise, and replace every inequality of the second kind by (i) $x < y$, if $k_1 = k_2$; (ii) **true**, if $k_1 < k_2$; and (iii) **false** otherwise.

Every equality occurring in φ is now of the form $x + k_1 = y + k_2$. Replace every such equality by $x = y$ if $k_1 = k_2$, and by **false** otherwise.

Every use of monadic predicates in φ now has the form $X(x+k)$, for $k \leq N-1$. Replace every such predicate by $X_k(x)$.

Finally, replace every occurrence of $\forall X\, \psi$ in φ by $\forall X_0 \forall X_1 \ldots \forall X_{N-1}\, \psi$. The resulting formula is the desired $\overline{\varphi}$. Note that $\overline{\varphi}$ does not mention the $+1$ function, and is therefore indeed a non-metric (i.e., purely order-theoretic) sentence in $\mathsf{MSO}(<)$. The following proposition is then clear.

Proposition 3.8 ([ORW09])**.** $\langle [0, N), <, +1, \mathbf{P}\rangle \models \varphi$ *iff* $\langle [0, 1), <, \mathbf{P}_0, \ldots, \mathbf{P}_{N-1}\rangle \models \overline{\varphi}$.

Following this translation, the $\mathsf{WMSO}(<)$ formula $\overline{\varphi_{\mathcal{A}}}$ describes the winning condition for a game where each time a transition occurs, Automaton makes a move and Pathfinder responds. In a McNaughton game defined by $\overline{\varphi_{\mathcal{A}}}$ over the structure $\langle [0, 1), <, \overline{\mathbf{P}}_1, \ldots, \overline{\mathbf{P}}_{N-1}\rangle$, one constructs each $\overline{X_i}$ simultaneously (so at time t we have already constructed each $\overline{X_i}$ on $[0, t)$). Thus the causality of this game does not correlate to causality of the acceptance game; we wish to construct all of $\overline{X_0}$ before any of $\overline{X_1}$. In order to restore this relationship, we construct instead a series of linked games in the following section.

3.5.3 A Series of McNaughton Games

We now show that the decision problem for Weak MSO McNaughton games over bounded intervals $[0, N)$ with the $+1$ relation is decidable by breaking the problem into a series of N linked McNaughton games which do not mention the $+1$ relation using Proposition 3.8. We then apply this result in the case when the game was generated from an alternating timed automaton's acceptance game to conclude the decidability of time-bounded language emptiness.

Theorem 3.9. *Given a* $\mathsf{WMSO}(<, +1)$ *formula* $\varphi(\overline{P}, \overline{X}, \overline{Y})$ *and a time domain* $[0, N)$*, it is decidable whether there exists an interpretation* $\overline{\mathbf{P}}$ *of* $\overline{P}$ *over* $[0, N)$ *such that Player I wins* $\mathbb{G}(\mathcal{M}, \varphi)$ *where* $\mathcal{M} = \langle [0, N), <, +1, \overline{\mathbf{P}} \rangle$*.*

Proof. To simplify notation, we deal with the special case when φ mentions only a single X and Y variable, along with a single parameter variable P.

We first apply the translation of Proposition 3.8 to $\varphi(P, X, Y)$ and produce the formula $\overline{\varphi}(\overline{P}, \overline{X}, \overline{Y})$ which is in $\mathsf{WMSO}(<)$ only. We then consider a sequence of McNaughton games $\mathbb{G}_0, \ldots, \mathbb{G}_{N-1}$ over $[0, 1)$ such that the variables X_j and Y_j for $0 \leq j < i$ are additionally treated as parameters in the game $\mathbb{G}_i$. Intuitively, each of these games represent one time unit of the game $\mathbb{G}(\varphi, \mathcal{M})$.

In the first such game $\mathbb{G}_{N-1}$, the winning condition is $\gamma_{N-1} = \overline{\varphi} \wedge \chi_{N-1}$ and we consider $X_0, \ldots, X_{N-2}$ and $Y_0, \ldots, Y_{N-2}$ as parameters in addition to the original parameter set $\overline{P}$. In this game, we require Player I to construct X_{N-1} whilst Player II constructs Y_{N-1}. The conjunct χ_{N-1} is used to enforce the requirement that Player I and Player II move only when the parameter P_{N-1} becomes true (as by the definition of the McNaughton game they may move when other parameters are true) and is defined as follows:

$$\chi_i \stackrel{\text{def}}{=} \forall t(X_i(t) \rightarrow P_i(t)) \vee \neg \forall t(Y_i(t) \rightarrow P_i(t))$$

We apply Proposition 3.6 to solve the Parameter Problem for this game, determining

from what parameter values Player I can be sure to win in the form of a new formula

$$\theta_{N-1}(\overline{P}, X_0, \ldots, X_{N-2}, Y_0, \ldots, Y_{N-2}) \ .$$

We then set γ_{N-2}, the winning condition of the game $\mathbb{G}_{N-2}$, as $\gamma_{N-2} = \theta_{N-1} \wedge \chi_{N-2}$ and iterate this process, applying Proposition 3.6 each time to obtain θ_i from γ_i.

This completes the definition of the games $\mathbb{G}_0, \ldots, \mathbb{G}_{N-1}$. There is a natural bijective correspondence between the set of positions of $\mathbb{G}(\mathcal{M}, \varphi)$ and the set of positions of (the various instantiations of) the $\mathbb{G}_i$, where a position of $\mathbb{G}(\mathcal{M}, \varphi)$ with timestamp t corresponds to a position of $\mathbb{G}_{\lfloor t \rfloor}$ with timestamp $t - \lfloor t \rfloor$. Moreover this association preserves the identity of the winning player. In particular, Player I wins $\mathbb{G}(\mathcal{M}, \varphi)$ if and only if he wins $\mathbb{G}(\overline{\mathcal{M}}, \gamma_0)$ where $\overline{\mathcal{M}} = \langle [0, 1), <, \overline{\mathbf{P}} \rangle$. We omit the details.

If we apply Proposition 3.6 one final time to $\mathbb{G}(\overline{\mathcal{M}}, \gamma_0)$, we obtain a formula $\theta_0(\overline{P})$ such that if θ_0 is satisfiable over $\langle [0, 1), < \rangle$, there exists an interpretation of the parameters $\overline{P}$ such that Player I has a winning strategy in $\mathbb{G}(\overline{\mathcal{M}}, \gamma_0)$ and hence in $\mathbb{G}(\mathcal{M}, \varphi)$. This completes the proof, since $\mathsf{WMSO}(<)$ satisfiability is decidable over bounded time domains [ORW09]. □

Applying Theorem 3.9 in the case that the game is the $\mathbb{G}(\mathcal{M}_w, \varphi_{\mathcal{A}})$ obtained from $G(\mathcal{A}, w)$ by Proposition 3.7, we obtain the main result of this chapter.

Theorem 3.10. *The time-bounded language emptiness problem for alternating timed automata is decidable.*

3.6 Hardness of Alternating Timed Automata

In this section, we show that the time-bounded language emptiness problem for alternating timed automata is non-elementary in the time bound. We achieve this by simulating counter machines with automata, proving that if the maximum size of the counters is bounded, it is possible to verify an arbitrarily long computation history of such a machine in bounded time. (The bound on the counter size increases with additional time units.)

3.6.1 Counter Machines

In [AD94] Alur and Dill show that universality for timed automata is undecidable by reduction from the recurrence problem for 2-counter machines. Their proof involves two steps: first they encode the set of computation histories of such a machine M as a timed language L_M; and second they show how to construct an automaton that recognises the complement of L_M (more details can be found in Section 2.3.5). This proof can straightforwardly be adapted to show undecidability of language emptiness for alternating timed automata by constructing an alternating timed automaton that accepts L_M. In this chapter, we use a variant of these machines with four counters which are each bounded by some maximum value K.

Definition 3.11. A *K-bounded 4-counter machine* M has a set of n locations and four K-bounded counters C, D, E and F. We represent a configuration of such a machine by a 5-tuple $\langle i, c, d, e, f \rangle$ where $1 \leq i \leq n$ and $0 \leq j \leq K$ for $j \in \{c, d, e, f\}$. Each location corresponds to an instruction to increment or decrement one of the counters, or a jump instruction which is conditional on one of the counters being 0. Jump instructions have two possible next instructions while non-jump instructions have one.

We note that these machines are able to simulate space-bounded Turing machines (see [Min61]).

Proposition 3.12. *Given a Turing machine T with space bound K and input x, one can construct a 2^K-bounded 4-counter machine $M(T, x)$ in space logarithmic in $|T|$ and $|x|$ such that $M(T, x)$ has an accepting computation if and only if T accepts x and $M(T, x)$ is of size linear in $|T|$ and $|x|$.*

3.6.2 Encoding Computation Histories

Alur and Dill's encoding of 2-counter machine computations as timed words in [AD94] involves encoding configurations as sequences of events in a single time interval, with successive configurations in successive time units. This idea is not appropriate to yield optimal lower bounds for the time-bounded emptiness problem for alternating timed automata. (A direct application would yield only an ExpTime lower bound.) Instead our idea is to choose a maximum value k and encode the counters as "decorated bit strings" at this level k. Such a bit string decorates each level k binary bit with a trailing "address", which is a level $k-1$ encoding. We repeat these decorated bit strings k times (repeating a sequence of events with identical fractional parts in k time units of a timed word) so that our alternating timed automaton can decide whether such a timed word represents a correct encoding; the automaton uses one time unit to verify each level of the encoding.

We define the family of $\exp_k$ functions (which exponentiate their argument k times) inductively by $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$, so

$$\exp_k(n) = 2^{\cdot^{\cdot^{\cdot^{2^n}}}\Big\}k}$$

We say that a function $f(n) : \mathbb{N} \to \mathbb{N}$ is *non-elementary* if it grows faster than $\exp_k(n)$ for any fixed k.

Let $\Sigma_k = \{0_i, 1_i \mid 1 \le i \le k\}$ be an alphabet. We show how to encode a number n in the range $0 \le n < \exp_k(1)$ as a string $r_k(n)$ in Σ_*. Let $x_0 x_1 \ldots x_{\exp_{k-1}(1)-1}$ be the binary representation of n, padded with leading zeros to length $\exp_{k-1}(1)$. Then $r_k(n)$ is defined inductively on k by

$$\begin{aligned} r_1(0) &= 0_1 \\ r_1(1) &= 1_1 \\ r_{k+1}(n) &= (x_0)_{k+1}\, r_k(0)(x_1)_{k+1}\, r_k(1) \ldots (x_{\exp_k(1)-1})_{k+1}\, r_k(\exp_k(1) - 1) \end{aligned}$$

We say that the $r_k(i)$ in this definition is the "address" of $(x_i)_{k+1}$. We illustrate this definition by encoding the value 9 with $k = 3$. Since 9 is 1001 in binary, we have:

$$\begin{aligned} r_3(11) &= 1_3\, r_2(0) 0_3\, r_2(1) 0_3\, r_2(2) 1_3\, r_2(3) \\ &= 1_3 0_2\, r_1(0) 0_2\, r_1(1) 0_3 0_2\, r_1(0) 1_2\, r_1(1) 0_3 1_2\, r_1(0) 0_2\, r_1(1) 1_3 1_2\, r_1(0) 1_2\, r_1(1) \\ &= 1_3 0_2 0_1 0_2 1_1 0_3 0_2 0_1 1_2 1_1 0_3 1_2 0_1 0_2 1_1 1_3 1_2 0_1 1_2 1_1 \end{aligned}$$

In the next section we wish to verify that a word in Σ_k^* is a correct encoding of a number, so we specify some useful predicates on words in Σ_k^* as follows:

- $\mathit{Encode}_k(u)$ holds if and only if there exists $i < \exp_k(1)$ such that $u = r_k(i)$.
- $\mathit{Eq}_k(u, v)$ holds if and only if $u = r_k(i) = v$ for some $i < \exp_k(1)$.
- $\mathit{First}_k(u)$ holds if and only if $u = r_k(0)$.
- $\mathit{Last}_k(u)$ holds if and only if $u = r_k(\exp_k(1) - 1)$.
- $\mathit{Succ}_k(u, v)$ holds if and only if there exists $i < \exp_k(1) - 1$ such that $u = r_k(i)$ and $v = r_k(i + 1)$.
- $\mathit{Pred}_k(u, v)$ holds if and only if $\mathit{Succ}_k(v, u)$ holds.

For $k = 1$, these predicates are all trivial to define and for higher values of k, we can recursively produce definitions of them in terms of predicates with lower values of k, as follows.

$\mathit{Encode}_k(u)$ holds if w has the form $u_0 w_0 u_1 w_1 \ldots u_n w_n$ where $u_i \in \{0_k, 1_k\}$, $w_i \in \Sigma_{k-1}^+$, $\mathit{First}_{k-1}(w_0)$ holds, $\mathit{Last}_{k-1}(w_n)$ holds and for each $i < \exp_k(1) - 1$, $\mathit{Succ}_{k-1}(w_i, w_{i+1})$ holds.

These conditions ensure that each bit is followed by an address, that the first bit is followed by the zero address $r_{k-1}(0)$, that the last bit is followed by the highest possible address $r_{k-1}(\exp_{k-1}(1) - 1)$ and that the address of each bit is one more than the address of the previous bit.

Observe that we can define $First_k(u)$ by modifying the above to require that $u_i = 0_k$ for all i and define $Last_k(u)$ by instead requiring that $u_i = 1_k$ for all i.

$Succ_k(u, v)$ holds if $Encode_k(u)$ holds, $Encode_k(v)$ holds, u has the form $u_0 w_0 u_1 w_1 \dots u_n w_n$ and v has the form $v_0 y_0 v_1 y_1 \dots v_m y_m$ for $u_i, v_i \in \{0_k, 1_k\}$ and $w_i, y_i \in \Sigma_{k-1}^+$ and there exists $s < \exp_k(1)$ such that $u_s = 0_k$, $v_s = 1_k$, for all $i < s$, $u_i = v_i$ and for all $i > s$, $u_i = 1_k$ and $v_i = 0_k$.

These conditions require that bit s is the most significant bit flipped by incrementing u to v; all the less significant bits of u must be 1 and of v must be 0 and all the more significant bits of u and v must be equal. Note that the definition of $Encode_k$ ensures that both u and v have exactly $\exp_k(1)$ level k bits, so $n = m$.

It is easy to define $Eq_k(v, w)$ and $Pred_k(v, w)$ in a similar way.

The configuration $\langle i, c, d, e, f\rangle$ of a 4-counter machine M can be represented by the word $conf_k(i, c, d) = b_i\, r_k(c) \#\, r_k(d) \#\, r_k(e) \#\, r_k(f)$, where b_i represents the location i, $r_k(c)$ represents c in the manner described above (and likewise $r_k(d)$ represents d, $r_k(e)$ represents e and $r_k(f)$ represents f); note that this is only meaningful if $\exp_k(1) > \max(c, d, e, f)$. The alphabet of such a representation is then $\Gamma_k = \{b_0, \dots, b_n\} \cup \Sigma_k \cup \{\#\}$. A finite computation history of M can be represented as the concatenation of the successive words which represent the configurations in the history. Thus the word

$$conf_k(i_0, c_0, d_0, e_0, f_0)\, conf_k(i_1, c_1, d_1, e_1, f_1) \dots conf_k(i_m, c_m, d_m, e_m, f_m)$$

represents an accepting computation of M if $\langle i_0, c_0, d_0, e_0, f_0\rangle$ is an initial configuration of M, $\langle i_m, c_m, d_m, e_m, f_m\rangle$ is an accepting configuration of M and for $0 \le j < m$, the transition $\langle i_j, c_j, d_j, e_j, f_j\rangle \to \langle i_{j+1}, c_{j+1}, d_{j+1}, e_{j+1}, f_{j+1}\rangle$ is possible in M. We thus define the (untimed) language $Acc_k(M) \subseteq \Gamma_k^*$ of words which represent an accepting computation of M with this value of k fixed. Note again that $Acc_k(M)$ will only be meaningful if $K < \exp_k(1)$ (as otherwise not all configurations of M can be represented by words in Γ_k^*).

We now construct a timed language L_M such that each timed word in L_M contains a

word w from $Acc_k(M)$ in the first time unit $[0, 1)$ and repeats w in each subsequent time unit $[i, i + 1)$ for $i + 1 < 2(k + 1)$, where k is the smallest integer such that $K < \exp_k(1)$. More precisely, L_M consists of those timed words $(\sigma_1, \tau_1), \ldots, (\sigma_m, \tau_m)$ where there exists $p \in \mathbb{N}$ such that:

1. $w = \sigma_1, \ldots, \sigma_p \in Acc_k(M)$.
2. $\tau_i < 1$ if and only if $i \leq p$.
3. For all $i \leq p$ and $j < 2(k + 1)$, $\sigma_i = \sigma_{i+jp}$ and $\tau_i = \tau_{i+jp} + j$.

Note that we crucially rely on the density of time to ensure that we can place all the timestamps of w in the first time unit.

Having constructed the timed language L_M, whose words represent accepting computation histories of M repeated in each of $2(k+1)$ time units, we will construct an alternating timed automaton whose language is L_M in the next section.

3.6.3 Simulating Counter Machines

Proposition 3.13. *Given k in unary and an $\exp_k(1)$-bounded 4-counter machine M, one can construct an alternating timed automaton $\mathcal{A}_M$ in space logarithmic in k and $|M|$ such that $L(\mathcal{A}_M) = L_M$.*

Proof. $\mathcal{A}_M$ is defined as the conjunction of the two alternating automata $\mathcal{A}_{copy}$ and $\mathcal{A}_{history}$. $\mathcal{A}_{copy}$ accepts the set of timed words that satisfy the third condition from Section 3.6.2 and is easily defined as a variant of the alternating timed automaton from Section 2.4.3.

Of the timed words that satisfy the third condition on L_M, $\mathcal{A}_{history}$ accepts only those timed words that further satisfy the first two conditions. In the following, we give a definition of $\mathcal{A}_{history}$.

The main challenge with verification of these properties is ensuring the correct succession of the various counter values in the computation history. Remembering one location

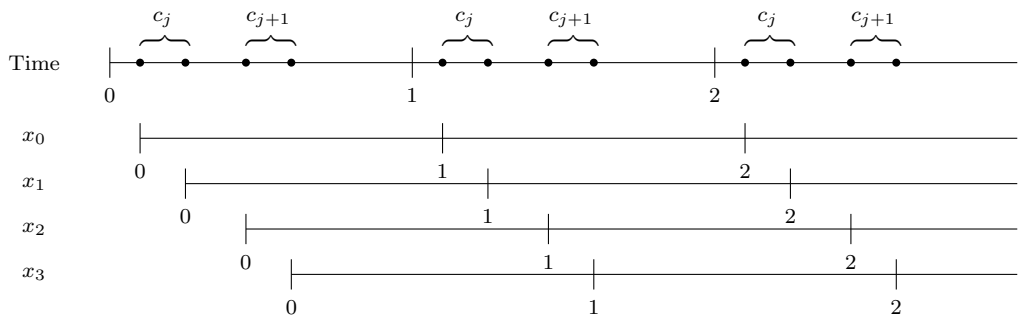


Figure 3.1: c_j and c_{j+1} captured by the clocks x_0, x_1, x_2 and x_3

of M in the state of $\mathcal{A}_{history}$ is easily possible, so we can easily check that if the subword seen in the first time unit encodes a sequence of configurations of M in the sense above, the successive location bits b_j, b_{j+1} correspond to transitions of M. The difficult part is then to ensure that the counter values c_j, d_j, e_j, f_j, c_{j+1}, d_{j+1} e_{j+1} and f_{j+1} form a correct succession. Depending on the value of b_j, we require one of $Eq_k(c_j, c_{j+1})$, $Succ_k(c_j, c_{j+1})$ or $Pred_k(c_j, c_{j+1})$ to hold (and similarly for the values of the D, E and F counters).

Thanks to the repetition of c_j and c_{j+1} at the same place in each time unit, we assume that we use the first time unit to reset four clocks x_0, x_1, x_2 and x_3 to delimit the timed events corresponding to c_j and c_{j+1} in the sense that x_0 has integer values at the first timed event in c_j, x_1 has integer values at the last timed event in c_j, x_2 has integer values at the first timed event in c_{j+1} and x_3 has integer values at the last timed event in c_{j+1}; this relationship is pictured in Figure 3.1.

We then define a component of $\mathcal{A}_{history}$ (a *gadget*) for each of $Succ_l$, $Pred_l$, Eq_l and each $l \leq k$ which uses these four clocks to detect its input; these gadgets are mutually recursive. We will also require gadgets corresponding to $Encode_l$, $First_l$ and $Last_l$ and assume that these gadgets use the two clocks x_0 and x_1 to detect their single input. We illustrate our strategy in the case of $Succ_l$; the other cases are similar.

In order to check $Succ_l(u, v)$, we first need to check $Encode_l(u)$ and $Encode_l(v)$ and use

conjunction to effectively run these checks in parallel with the rest of our verification. We thus have a gadget as in Figure 3.2 that passes both u and v to the $Encode_l$ gadget by resetting the clocks x_0 and x_1 when the delimiters of u or v have value 1; in the case of u the conditions $x_0 = 1$ and $x_1 = 1$ are used and in the case of v, the conditions $x_2 = 1$ and $x_3 = 1$ are used instead.

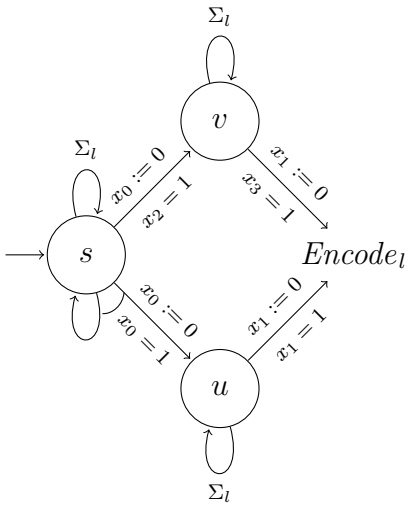


Figure 3.2: Gadget to separate two words to check $Encode_l$

Assuming that $Encode_l(u)$ and $Encode_l(v)$ both accept, u has the form $u_0w_0u_1w_1 \dots u_mw_m$ and v has the form $v_0y_0v_1y_1 \dots v_my_m$ for $u_i, v_i \in \{0_l, 1_l\}$ and $w_i, y_i \in \Sigma_{l-1}^+$. We must now check that there exists s such that $u_s = 0_l$, $v_s = 1_l$, for all $i < s$, $u_i = v_i$ and for all $i > s$, $u_i = 1_l$ and $v_i = 0_l$.

We therefore build a gadget, illustrated in Figure 3.3, which guesses a value for s and conjunctively verifies that for each bit $u_i \in \{0_l, 1_l\}$ in u the corresponding bit v_i is correct. In order to do this, the gadget marks the address w_i of u_i with resets of the clocks x_0 and x_1, then guesses which bit v_j has $j = i$, verifies that either $u_i = v_j$, $u_i = 0_l$ and $v_j = 1_l$ or $u_i = 1_l$ and $v_j = 0_l$ as appropriate and marks the address y_j with resets of the clocks x_2 and x_3. It then remains to check that the guess that $i = j$ is correct by checking $Eq_{l-1}(w_i, y_j)$; the clocks are set exactly as required to run the gadget Eq_{l-1} in the next time unit.

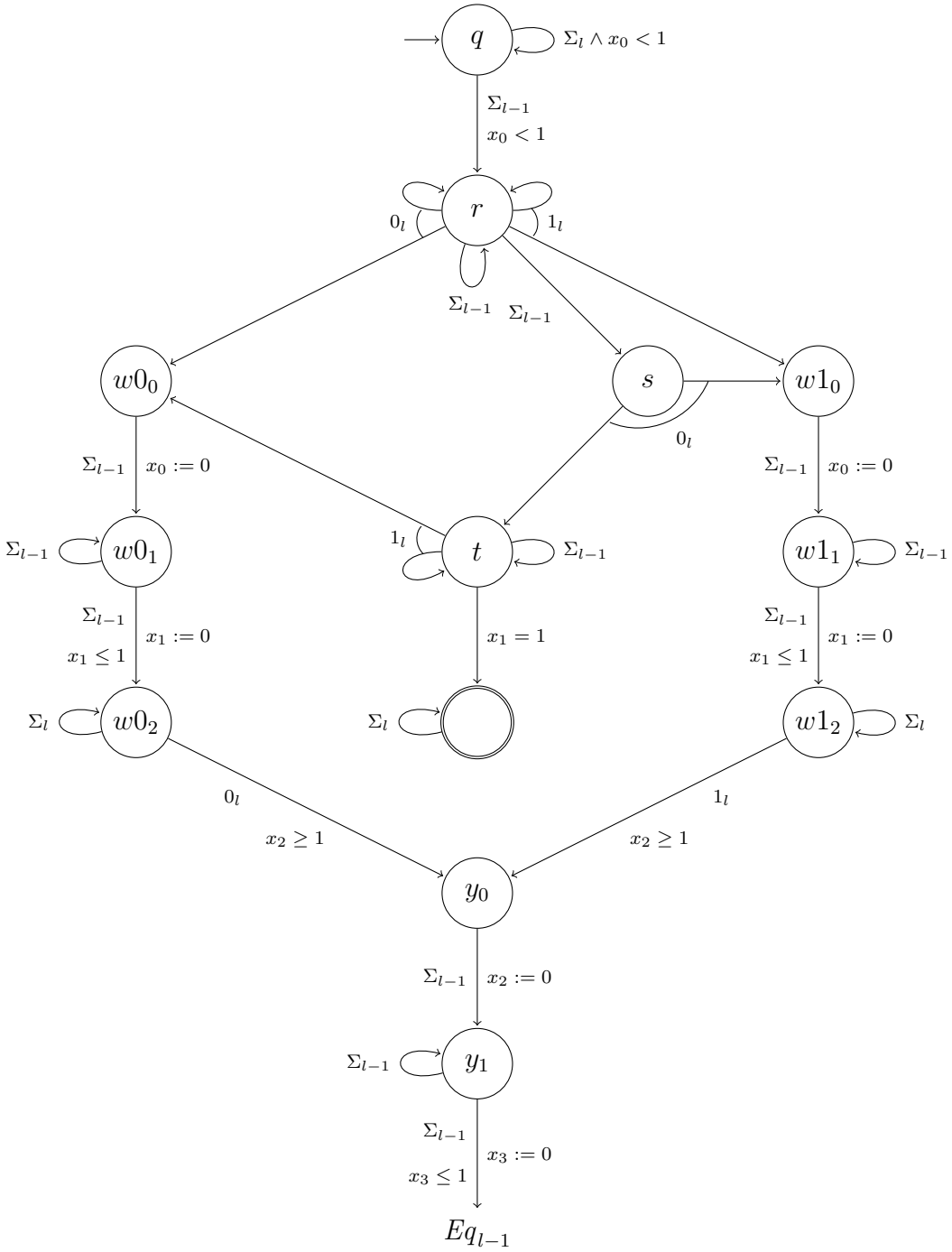


Figure 3.3: Gadget to partially test $Succ_l$

The gadget of Figure 3.3 takes one time unit to reduce its part of the verification of $Succ_l$ to instances of Eq_{l-1} and the gadget of Figure 3.2 takes one time unit to reduce its part of the verification of $Succ_l$ to instances of $Encode_l$. In a similar fashion, we can define gadgets which take one time unit to reduce $Encode_l$ to instances of $Succ_{l-1}$, $First_{l-1}$ and $Last_{l-1}$.

It thus takes $\mathcal{A}_{history}$ two time units to reduce all instances of verification tasks at level l to tasks at level $l-1$ and hence fewer than $2(k+1)$ time units in total (accounting for an initial time unit to decide which level k tasks need to be performed). Hence $\mathcal{A}_M$ can be sure to accept exactly words in L_M.

This construction requires $O(k)$ distinct gadgets which can be constructed using a single counter to a position of k assuming it is encoded in unary. Moreover the size of $\mathcal{A}_M$'s alphabet is linear in the size of the alphabet of M, so the transitions of these gadgets can be constructed using in addition a fixed number of pointers to elements of M. Further, the embedded copy of the locations of M required by $\mathcal{A}_M$ to ensure that the b_i letters form a correct succession can be constructed using a fixed number of pointers to elements of M. Hence we conclude that the construction of $\mathcal{A}_M$ requires only space logarithmic in k and $|M|$ (assuming k is encoded in unary). □

For a given time bound $2k$ and size $n \geq 2k$, we can define a space bound $K = \exp_k(1)$ such that for any Turing machine T of size n with space bound K and input x, one can use Proposition 3.12 to construct a 2^K-bounded 4-counter machine $M(T,x)$ such that $M(T,x)$ has an accepting computation if and only if T accepts x. One can then use Proposition 3.13 to construct an alternating timed automaton $\mathcal{A}$ which accepts exactly the $2k$-bounded language $L_{M(T,x)}$. Hence $\mathcal{A}$ accepts some $2k$-bounded word if and only if T accepts x. Composing these two reductions can be achieved in space logarithmic in n, $|x|$ and k, so we conclude the following theorem.

Theorem 3.14. *The time-bounded language emptiness problem for alternating timed automata with time bound k and size $n \geq k$ is hard for* $\textsc{Space}(\exp_k(1))$.

3.7 Summary

In this chapter we introduced the time-bounded language emptiness problem for alternating timed automata and gave a decision procedure which reduces this problem to solving a certain kind of logical game over the reals. We presented this game as an extension of the classical notion of McNaughton games — our extended games make use of parameters in order to model the word input to an alternating timed automaton. We avoid several difficulties by using the time points when these parameters become true to define the rounds of the game.

We computed the set of parameters for which a player can win such a game over the reals where the winning condition mentions only the $<$ relation and used this result to build a series of games which allow us to compute the winner of a game where the winning condition mentions the $+1$ relation in addition to $<$. This provided the decision procedure for the time-bounded language emptiness problem but suffered an exponential blow-up for each time unit.

We proved that this blow-up is unavoidable by exhibiting a reduction from the acceptance problem for 2-counter machines with bounded counters to the time-bounded language emptiness problem where the bound on the size of the counters is exponentially larger for each time unit needed.

Chapter 4

Real Time Uniformization

4.1 Introduction

Church's *synthesis problem* [Chu57] is to automatically construct an implementation of a given specification relating the inputs and outputs of a state-based system. The specification is assumed to be an $\mathsf{MSO}(<)$ formula $S(I, O)$, which determines a binary relation between input strings I and output strings O. An implementation is a function (or operator) P from strings to strings that *uniformizes* S in the sense that $S(I, P(I))$ holds for all inputs I. Church required that P be computable by a finite-state machine that at every moment $t \in \mathbb{N}$ reads an input symbol $I(t)$ and produces an output symbol $O(t)$. Hence, the output $O(t)$ produced at t depends only on input symbols $I(0), I(1), \ldots, I(t)$ received before t, that is, P should be a *causal* operator. In light of Büchi's proof [Büc62] of the expressive equivalence of $\mathsf{MSO}(<)$ and finite automata, P is finite-state if and only if it is $\mathsf{MSO}(<)$-definable. Church's synthesis problem can therefore be stated formally as follows.

Definition 4.1 (Church Synthesis Problem)**.** Given an $\mathsf{MSO}(<)$ formula $\varphi(X, Y)$, decide whether there is an MSO-definable causal operator F such that $\langle \mathbb{N}, < \rangle \models \forall X \varphi(X, F(X))$ and if so, construct this operator.

This problem, which is more general than the satisfiability problem for MSO over $\langle \mathbb{N}, < \rangle$, was shown decidable in a landmark paper of Büchi and Landweber [BL69]. Their main theorem is stated as follows:

Theorem 4.2 (Büchi and Landweber)**.** *Given an* MSO$(<)$ *formula* $\varphi(X, Y)$*, one can decide whether there is a causal operator that uniformizes* φ*. If such an operator exists then it can be represented by a finite-state automaton which can be computed from* φ*.*

Note that this theorem obviates one of Church's requirements: it guarantees that whenever φ has a uniformizer then it has a uniformizer that is computable by a finite-state automaton (equivalently, definable in MSO$(<)$).

In the continuous-time setting, one can naturally consider the synthesis problem over the non-negative reals rather than the naturals. Here we think of a specification as a relation between *signals* rather than words. As specification language one again takes MSO$(<)$, which has a natural interpretation over the non-negative reals. As implementations one again takes MSO$(<)$-definable causal operators.

Shelah [She75] proved that satisfiability for MSO$(<)$ is undecidable over the reals if we allow quantification over arbitrary predicates. In Computer Science however, it is natural to restrict to *finitely variable* predicates, that is, predicates whose characteristic function has finitely many discontinuities in any bounded interval. Under the finite-variability interpretation, MSO satisfiability is decidable over the reals and formulae have the same expressive power as finite-state automata [Rab02].

However, the full extension of the Büchi and Landweber theorem fails over the non-negative reals, even under the finite-variability assumption. For example, the formula that says that Y has at least two points of discontinuity can be uniformized by a causal operator, but not by an MSO$(<)$*-definable causal operator* (see Proposition 4.12 for details).

Nevertheless, we are able to show the following result:

Theorem 4.3. *Given an* $\mathsf{MSO}(<)$ *formula* $\varphi(\overline{X}, \overline{Y})$*, one can decide whether there is an* $\mathsf{MSO}(<)$*-definable causal operator that uniformizes* φ *over* $\langle \mathbb{R}_+, < \rangle$*. If such an operator exists, the algorithm computes a formula that represents the operator.*

In the continuous setting a deficiency of $\mathsf{MSO}(<)$ is that it cannot express *metric* properties such as "the distance between two points is one". Thus we consider specifications expressed in $\mathsf{MSO}(<, +1)$, which extends $\mathsf{MSO}(<)$ with the $+1$ function.

Unfortunately, even with the finitely-variable interpretation, the satisfiability problem over the non-negative reals is undecidable for $\mathsf{MSO}(<, +1)$ [HR04]. However in [ORW09] it was proved that $\mathsf{MSO}(<, +1)$ satisfiability is decidable for every fixed bounded-length interval of the reals. The main result of this chapter is that Church's synthesis problem is also decidable for $\mathsf{MSO}(<, +1)$ for every fixed bounded-length interval of the reals. Specifically,

Theorem 4.4. *Given an* $\mathsf{MSO}(<, +1)$ *formula* $\varphi(\overline{X}, \overline{Y})$ *and* $N \in \mathbb{N}$*, one can decide whether there is an* $\mathsf{MSO}(<, +1)$*-definable causal operator that uniformizes* φ *over the the interval* $[0, N)$*. If such an operator exists, the algorithm computes a formula that represents the operator.*

In order to prove Theorem 4.4 we need to consider the Church synthesis problem with *parameters*—additional predicates that the specification may reference but which do not have to be considered in a causal way by the implementation. This problem was considered in [Rab07a] for $\mathsf{MSO}(<)$ over $\langle \mathbb{N}, < \rangle$. Here we extend the Church synthesis problem with parameters to the non-negative reals; unlike [Rab07a] however, parameters are here used merely as a device to enable an inductive proof of our main result — they are not our primary object of study.

Finally, we show that the synthesis problem over bounded intervals of reals is non-elementary, even for specifications expressed in fragments of $\mathsf{MSO}(<, +1)$ with an elementary satisfiability problem over bounded intervals, such as MTL [ORW09].

4.2 Monadic Second-Order Logic

Recall from Section 2.6 that we consider $\mathsf{MSO}(<, +1)$ over a signature consisting of the binary relations $<$ and $+1$ and a countable family of monadic predicate names $P_0, P_1, \ldots$ and denote by $\mathsf{MSO}(<)$ the sub-language consisting of all formulae that do not mention the $+1$ relation.

We say that a subset $\mathbf{P} \subseteq \mathbb{R}_+$ is *finitely variable* if its characteristic function has finitely many discontinuities in any bounded sub-interval of $\mathbb{R}_+$. In this chapter, we consider a semantics for $\mathsf{MSO}(<, +1)$ where we restrict interpretations of monadic predicate names and variables to be finitely variable. This is an essential restriction: it is known that allowing unrestricted second-order quantification leads to an undecidable satisfiability problem [She75]. In case the domain $\mathbb{T}$ is unbounded we also make the harmless simplifying assumption that at least one of the predicates in any structure $\langle \mathbb{T}, <, \overline{\mathbf{P}} \rangle$ is not eventually constant. We later consider the discontinuities in such a structure as defining a word — this requirement means that every infinite structure defines an infinite word, simplifying our proofs as finite-word cases need not be considered.

Let $\Sigma_P = \{0,1\}^m$ be a finite alphabet. A structure $\langle \mathbb{T}, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ corresponds to a function $f : \mathbb{T} \to \Sigma_P$, where $f(t)_i = 1$ if $t \in \mathbf{P}_i$ and $f(t)_i = 0$ otherwise. If $\mathbb{T} \subseteq \mathbb{R}_+$ then f is called a *signal* and if $\mathbb{T} \subseteq \mathbb{N}$ then f is a (finite or infinite) word. By assumption, a signal is finitely variable and if its domain is unbounded it is not eventually constant. We denote by Σ^ω the set of all infinite words over an alphabet Σ and by $Sig(\Sigma)$ the set of all signals over Σ with domain $\mathbb{R}_+$. An $\mathsf{MSO}(<)$ sentence $\varphi(\overline{P})$ that mentions predicate names $P_1, \ldots, P_m$ respectively defines a word language $L_{\mathbb{N}}(\varphi) \stackrel{\text{def}}{=} \{w \in (\Sigma_P)^\omega : w \models \varphi\}$ and a signal language $L_{\mathbb{R}}(\varphi) \stackrel{\text{def}}{=} \{f \in Sig(\Sigma_P) : f \models \varphi\}$.

4.3 Transforming Between Words and Signals

In this chapter we answer questions about MSO over the non-negative reals under the finite variability interpretation by reducing them to questions about MSO over the naturals. This section presents the foundations of this reduction—semantic translations between signal languages and word languages and corresponding syntactic translations on MSO(<) formulae. We concentrate here on signals with domain $\mathbb{R}_+$ and on infinite words, though the ideas easily apply to signals with bounded domain and to finite words.

Let $f : \mathbb{R}_+ \to \Sigma$ be a signal over alphabet Σ. Recall that by assumption f has a countably infinite and unbounded set of discontinuities. Define a *sampling sequence* for f to be an unbounded strictly increasing sequence of reals $0 = \tau_0 < \tau_1 < \tau_2 < \ldots$ that includes all discontinuities of f within its even positions (i.e. if t is a discontinuity of f, there exists i such that $t = \tau_{2i}$). Given a sampling sequence τ we define the word $W_\tau(f) \in \Sigma^\omega$ by $W_\tau(f) = f(\tau_0)f(\tau_1)f(\tau_2)\ldots$. Given a language $L \subseteq Sig(\Sigma)$ we define the corresponding word language $L^\dagger \subseteq \Sigma^\omega$ to comprise all words $W_\tau(f)$ where $f \in L$ and τ is a sampling sequence for f. As we now explain, the language $L^\dagger$ is *stutter-closed*.

Say that a word $a = a_0a_1\ldots$ is *stutter-free* if for all i either $a_{2i-1} \neq a_{2i}$ or $a_{2i} \neq a_{2i+1}$ (i.e. no letter in an odd position is repeated in the next two positions). If $a_k = a_{k+1} = a_{k+2}$ for some odd k, we say that the word *stutters at* k. Let $sf(a) = a_{i_0}a_{i_1}\ldots$ where $i_0 = 0$, $i_{2l+1} = i_{2l} + 1$ and $i_{2l} = \max\{2k \mid \forall j(i_{2l-1} \leq j < 2k \to a_j = a_{i_{2l-1}})\}$. We define the relation $\sim$ of *stutter equivalence* on Σ^ω by $a \sim b$ if and only if $sf(a) = sf(b)$. It is straightforward that $sf(a)$ is the unique stutter-free word which is stutter equivalent to a. A language $L \subseteq \Sigma^\omega$ is *stutter-closed* if it saturates $\sim$. It is straightforward that if L is a signal language then the corresponding word language $L^\dagger$ is stutter-closed.

Define the *stutter-closure* of a language $L \subseteq \Sigma^\omega$ to be the smallest stutter-closed language $SC(L)$ that contains L. It is straightforward that $SC(L)$ is ω-regular if L is ω-regular.

Given a word $w = w_0w_1w_2\ldots \in \Sigma^\omega$ and an unbounded strictly increasing sequence of

reals $0 = \tau_0 < \tau_1 < \tau_2 < \ldots$, define the signal $S_\tau(w) : \mathbb{R}_+ \to \Sigma$ by

$$S_\tau(t) = \begin{cases} w_{2k} & t = \tau_{2k} \\ w_{2k+1} & \tau_{2k} < t < \tau_{2k+2} \end{cases}$$

Given a word language $L \subseteq \Sigma^\omega$, the corresponding signal language $L^\star$ comprises all signals of the form $S_\tau(w)$ for some $w \in L$ and unbounded sequence of reals τ. Having made these definitions of operators S_τ and W_τ, we observe that they are inverse mappings.

Lemma 4.5.

1. *If τ is a sampling sequence for a signal f, then $S_\tau(W_\tau(f)) = f$.*

2. *For any unbounded sequence of reals τ and word w, $W_\tau(S_\tau(w)) = w$.*

Proof of 1. Consider the point τ_{2i} for any i. Here $S_\tau(W_\tau(f))(\tau_{2i}) = W_\tau(f)_{2i} = f(\tau_{2i})$, as required. Alternatively, consider $t \neq \tau_{2i}$ for any i. Since τ is an unbounded set, there exists j such that $\tau_{2j} < t < \tau_{2j+2}$. Then

$$S_\tau(W_\tau(f))(t) = W_\tau(f)_{2j+1} = f(\tau_{2j+1}) = f(t)\,.$$

The last equivalence holds because f is continuous on (τ_{2j}, τ_{2j+2}). □

Proof of 2. For any i,

$$W_\tau(S_\tau(\sigma))_{2i} = S_\tau(\sigma)(\tau_{2i}) = \sigma_{2i} \text{ and}$$
$$W_\tau(S_\tau(\sigma))_{2i+1} = S_\tau(\sigma)(\tau_{2i+1}) = \sigma_{2i+1}$$

as required. □

Define the relation $\sim$ of *stretching equivalence* on the set $Sig(\Sigma)$ of signals over alphabet Σ by $f \sim g$ if and only if $f = g \circ \rho$ for some order isomorphism $\rho : \mathrm{dom}(f) \to \mathrm{dom}(g)$. A signal language L is *speed-independent* if it saturates $\sim$. It is straightforward that $L_{\mathbb{R}}(\varphi)$ is

speed-independent for any $\mathsf{MSO}(<)$ formula φ. It is also clear that $L^\star$ is speed-independent for any word language $L \subseteq \Sigma^\omega$.

The operators $(-)^\dagger$ and $(-)^\star$ define a bijection between stutter-closed word languages and speed-independent signal languages:

Proposition 4.6. *If $L \subseteq \Sigma^\omega$ is stutter-closed then $L^{\star\dagger} = L$. If $L \subseteq Sig(\Sigma)$ is speed-independent then $L^{\dagger\star} = L$.*

We first prove two technical lemmas concerning the functions W and S.

Lemma 4.7. *For any word w and unbounded sequences of reals τ and τ', $S_\tau(w) \sim S_{\tau'}(w)$.*

Proof. Let $\rho : \mathbb{R}_+ \to \mathbb{R}_+$ be a piecewise linear order isomorphism that maps 0 to 0 and τ_{2i} to τ'_{2i}. Then

$$[S_{\tau'}(w) \circ \rho](\tau_{2k}) = S_{\tau'}(w)(\tau'_{2k}) = w_{2k} = S_\tau(w)(\tau_{2k}) \text{ and}$$
$$[S_{\tau'}(w) \circ \rho](t) = S_{\tau'}(w)(\rho(t)) = w_{2k+1} = S_\tau(w)(t) \text{ for } \tau_{2k} < t < \tau_{2k+2}$$

Hence $[S_{\tau'}(w) \circ \rho] = S_\tau(w)$ so $S_{\tau'}(w) \sim S_\tau(w)$. □

Lemma 4.8. *For any signal f and sampling sequences τ and τ', $W_\tau(f) \sim W_{\tau'}(f)$.*

Proof. Let λ be the set of discontinuities of f. Then since τ and τ' are sampling sequences for f, let k_i be such that $\tau_{k_i} = \lambda_i$ and l_i be such that $\tau'_{l_i} = \lambda_i$. Note that then k_i and l_i are even for all i. $W_\tau(f)_j$ is constant for $k_i < j < k_{i+1}$ and either $W_\tau(f)_{k_i-1} \neq W_\tau(f)_{k_i}$ or $W_\tau(f)_{k_i} \neq W_\tau(f)_{k_i+1}$ by definition of discontinuities. Hence

$$sf(W_\tau(f)) = W_\tau(f)_{k_0} W_\tau(f)_{k_0+1} W_\tau(f)_{k_1} \ldots$$
$$\text{and similarly } sf(W_{\tau'}(f)) = W_{\tau'}(f)_{l_0} W_{\tau'}(f)_{l_0+1} W_{\tau'}(f)_{l_1} \ldots$$

Then $W_\tau(f)_{k_i} = f(\lambda_i) = W_{\tau'}(f)_{l_i}$ and $W_\tau(f)_{k_i+1} = f(\frac{1}{2}(\lambda_i + \lambda_{i+1})) = W_{\tau'}(f)_{l_i+1}$ hence $sf(W_\tau(f)) = sf(W_{\tau'}(f))$ so $W_\tau(f) \sim W_{\tau'}(f)$. □

We now prove the bijection proposition.

Proof of Proposition 4.6. Let $L \subseteq Sig(\Sigma)$ be a speed-independent language of signals. Clearly $L \subseteq L^{\dagger\star}$ as for any $f \in L$ and sampling sequence τ for f we have $W_\tau(f) \in L^\dagger$ and $f = S_\tau(W_\tau(f)) \in L^{\dagger\star}$.

Suppose that $f \in L^{\dagger\star}$. Then there exist w, τ such that $w \in L^\dagger$ and $f = S_\tau(w)$. Moreover, there exist g, τ' such that $g \in L$ and $w = W_{\tau'}(g)$. Then $S_{\tau'}(w) = g$ and by Lemma 4.7 we have $S_\tau(w) \sim S_{\tau'}(w)$, i.e. $f \sim g$. Since L is speed-independent and $g \in L$, $f \in L$ so $L^{\dagger\star} = L$.

Now consider a stutter-closed language of words $L \subseteq \Sigma^\omega$. Again $L \subseteq L^{\star\dagger}$ is clear since for any $w \in L$, $S_{\mathbb{N}}(w) \in L^\star$ and $w = W_{\mathbb{N}}(S_{\mathbb{N}}(w)) \in L^{\star\dagger}$

Suppose that $w \in L^{\star\dagger}$. Then there exist f, τ such that $f \in L^\star$ and $w = W_\tau(f)$. Moreover, there exist u, τ' such that $u \in L$ and $f = S_{\tau'}(u)$. Then $W_{\tau'}(f) = u$ and by Lemma 4.8 we have $W_\tau(f) \sim W_{\tau'}(f)$, i.e. $w \sim u$. Since L is stutter-closed and $u \in L$, $w \in L$ so $L^{\star\dagger} = L$. □

Next we recall from [Rab02] syntactic analogs of the language operators $(-)^\dagger$ and $(-)^\star$. The following results show that the language operators $(-)^\dagger$ and $(-)^\star$ preserve MSO-definability.

Proposition 4.9. *Given an* MSO$(<)$ *sentence* $\varphi(\overline{P})$*, one can compute another* MSO$(<)$ *sentence* $\varphi^\dagger(\overline{P})$ *such that* $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.

The definition of $\varphi^\dagger$ depends crucially on our assumption that signals are finitely variable. This allows us to reduce an MSO statement about a signal to a statement about the ω-sequence of values that the signal assumes.

Proposition 4.10. *Given an* MSO$(<)$ *sentence* $\varphi(\overline{P})$*, one can compute another* MSO$(<)$ *sentence* $\varphi^\star(\overline{P})$ *such that* $L_{\mathbb{R}}(\varphi^\star) = L_{\mathbb{N}}(\varphi)^\star$.

Some details of the proofs of these propositions are omitted in the literature so we expand and clarify the proofs of [Rab02] in Appendix A.

Finally, we note that the transformations $(-)^\star$ and $(-)^\dagger$ are inverse.

Lemma 4.11. *For any stutter-closed* MSO($<$) *sentence* φ, $L_{\mathbb{R}}(\varphi^\star)^\dagger = L_{\mathbb{N}}(\varphi)$. *For any* MSO($<$) *sentence* φ, $L_{\mathbb{N}}(\varphi^\dagger)^\star = L_{\mathbb{R}}(\varphi)$.

Proof. We have $L_{\mathbb{R}}(\varphi^\star) = L_{\mathbb{N}}(\varphi)^\star$ by Proposition 4.10. Then $L_{\mathbb{R}}(\varphi^\star)^\dagger = L_{\mathbb{N}}(\varphi)^{\star\dagger} = L_{\mathbb{N}}(\varphi)$ by Proposition 4.6 since $L_{\mathbb{N}}(\varphi)$ is stutter-closed.

By Proposition 4.9, we know that $L_{\mathbb{N}}(\varphi^\dagger) = L_{\mathbb{R}}(\varphi)^\dagger$. Then $L_{\mathbb{N}}(\varphi^\dagger)^\star = L_{\mathbb{R}}(\varphi)^{\dagger\star} = L_{\mathbb{R}}(\varphi)$ by Proposition 4.6 since $L_{\mathbb{R}}(\varphi)$ is speed-independent. □

4.4 Church's Problem with Parameters

Recall that a binary relation R is *uniformized* by a partial function f if $f \subseteq R$ and $\mathrm{dom}(f) = \mathrm{dom}(R)$. In this chapter we are interested in uniformizing MSO-definable relations by MSO-definable functions. The problem of uniformizing MSO-definable relations on the structure $\langle \mathbb{N}, < \rangle$ was first studied over fifty years ago by Church [Chu57], motivated by the problem of synthesising circuits from relational input-output specifications. Later Rabinovich [Rab07a] and Hänsch, Slaats and Thomas [HST09] considered the problem of uniformization over *labelled chains* $\langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$.

Our eventual goal is to study uniformization of MSO-definable relations over the structure $\langle \mathbb{T}, <, +1 \rangle$ for $\mathbb{T}$ a bounded interval of reals. We delay a treatment of the $+1$ relation until the following section. Here we lay the groundwork by considering uniformization of labelled chains $\langle \mathbb{R}_+, <, \overline{\mathbf{P}} \rangle$. This extends the treatment of the labelled case from [HST09, Rab07a] to dense orders.

4.4.1 The Uniformization Problem

Consider a second-order language over a signature including the binary relation symbol $<$ and monadic predicate names $P_1, \ldots, P_m$. Let $\mathcal{M} = \langle \mathbb{T}, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ be a labelled chain. We say that an MSO formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ uniformizes an MSO formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over $\mathcal{M}$ if $\mathcal{M}$ satisfies the following sentences:

1. $\forall \overline{X} \forall \overline{Y} \left(\psi(\overline{P}, \overline{X}, \overline{Y}) \rightarrow \varphi(\overline{P}, \overline{X}, \overline{Y}) \right)$

2. $\forall \overline{X} \exists^{=1} \overline{Y} \psi(\overline{P}, \overline{X}, \overline{Y})$

We call the predicate names $\overline{P}$ *parameters* and $\overline{X}, \overline{Y}$ *variables*. We say that $\psi(\overline{P}, \overline{X}, \overline{Y})$ uniformizes $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over a class of chains $\mathcal{C}$ if $\psi(\overline{P}, \overline{X}, \overline{Y})$ uniformizes $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over each individual chain in $\mathcal{C}$. Recall that $\exists^{=1} \overline{Y}$ means that there exists exactly one vector $\overline{Y}$ which satisfies the condition and notice that the above conditions can only hold if $\varphi(\overline{P}, \overline{X}, \overline{Y})$ is a total relation. There is, however, no loss of generality in considering only uniformization for total relations.

We say that a formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ satisfying 1 above is *faithful* to φ and a formula satisfying 2 above is *functional*. We furthermore say that ψ is *causal* if the following sentence holds in $\mathcal{M}$:

3. $\forall \overline{X}\, \overline{Y}\, \overline{U}\, \overline{V}\, \forall t \left[\psi(\overline{P}, \overline{X}, \overline{Y}) \wedge \psi(\overline{P}, \overline{U}, \overline{V}) \wedge \left(\forall s \leq t \, (\overline{X}(s) = \overline{U}(s)) \right) \Rightarrow \overline{Y}(t) = \overline{V}(t) \right]$

Intuitively a function is causal if its output at any time only depends on its input in the past—a reasonable assumption for any realisable function.

Roughly speaking, the uniformization problem is to determine whether a given formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ has a uniformizer over a given structure, or class of structures, and if so to compute such a uniformizer.

In Chapter 3, we chose to give a direct treatment of the language emptiness problem for alternating timed automata in terms of McNaughton games. We note that, using the MSO

formula from Section 3.5.1, one can give a reduction of the language emptiness problem to the uniformization problem. However the latter problem is much more general and, as we will see, its solution involves many technical issues not present in Chapter 3 (in particular, those pertaining to stuttering).

We are interested here in uniformizers that are definable by an MSO formula and this proves an important restriction over structures with real-valued domains. The following example illustrates a case where one can easily think of a uniformizer for φ, but no such uniformizer can be definable in MSO.

Proposition 4.12. *There is a formula $\varphi(X, Y)$ (even without parameters) such that there is a causal operator which uniformizes φ over the reals, however no* MSO*-definable causal operator uniformizes it.*

Proof. Let $\varphi(X, Y)$ be an $\mathsf{MSO}(<)$ formula which says that Y has at least two points of discontinuity. It is clear that the operator F which ignores its input and sets $F(X)(t) = 1$ if and only if $t \in ([0, 1) \cup [2, 3))$ uniformizes this formula, however we prove below there is no MSO-definable uniformizer $\psi(X, Y)$.

Let $\psi(X, Y)$ be an $\mathsf{MSO}(<)$ formula that defines a functional operator. Interpret X by the constant false signal $\mathbf{X}$ and let $\mathbf{Y}$ be the unique interpretation of Y such that $\psi(\mathbf{X}, \mathbf{Y})$ holds. For any order isomorphism $\rho : \mathbb{R}_+ \to \mathbb{R}_+$, $\psi(\mathbf{X} \circ \rho, \mathbf{Y} \circ \rho)$ also holds, since every $\mathsf{MSO}(<)$ formula is speed-independent. But $\mathbf{X} \circ \rho = \mathbf{X}$; as ψ is functional we must also have $\mathbf{Y} \circ \rho = \mathbf{Y}$. It is easy to see that this entails that $\mathbf{Y}$ be constant, contrary to the requirement that it have two discontinuities. □

Motivated by failures such as this, we seek to compute the set of parameter values for which there exists a definable uniformizer along with a single formula which defines a uniformizer for all such parameter values. We formally state the main result of this section as follows:

Theorem 4.3. *Given an* $\mathsf{MSO}(<)$ *formula* $\varphi(\overline{P}, \overline{X}, \overline{Y})$*, one can compute a sentence* $\theta(\overline{P})$ *and formula* $\psi(\overline{P}, \overline{X}, \overline{Y})$ *such that for every structure* $\mathcal{M} = \langle \mathbb{R}_+, <, \overline{\mathbf{P}} \rangle$*,* φ *has an* $\mathsf{MSO}(<)$*-definable causal uniformizer over* $\mathcal{M}$ *if and only if* $\mathcal{M} \models \theta$ *and in this case* ψ *is such a causal uniformizer.*

In Theorem 4.3 we call φ the *winning condition*, ψ the *uniformizer* and θ the *domain formula*. We give a proof of Theorem 4.3 in Sections 4.4.2 and 4.4.3.

4.4.2 From Signals to Words

Let L_1 be a speed-independent language of signals over alphabet $\Sigma_P = \{0,1\}^m$ and let $L_2 = (L_1)^\dagger$ be the corresponding stutter-closed language of words. We identify a signal f with the corresponding structure $\langle \mathbb{R}_+, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$, where f is the characteristic function of $\overline{\mathbf{P}}$; we similarly identify a word w with the corresponding structure $\langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$. We further identify a language L with the class of structures that correspond to the elements of L.

We reduce the problem of computing a uniformizer of an $\mathsf{MSO}(<)$ formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over the class of signals L_1 to the problem of computing a uniformizer for the corresponding formula $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ over the class of words L_2. Superficially these two problems are quite different since L_1 is a class of dense orders and L_2 a class of discrete orders. The key fact that makes this reduction work is that for each signal $f \in L_1$ we include in L_2 the whole stutter-closed class of words representing f. Given this, the reduction is simply stated:

Theorem 4.13. *If* $\psi(\overline{P}, \overline{X}, \overline{Y})$ *is a causal uniformizer for* $\varphi(\overline{P}, \overline{X}, \overline{Y})$ *over* L_1*, then* $\psi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ *is a stutter-closed causal uniformizer for* $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ *over* L_2*. Conversely if* $\psi(\overline{P}, \overline{X}, \overline{Y})$ *is a stutter-closed causal uniformizer for* $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ *over* L_2*, then* $\psi^\star(\overline{P}, \overline{X}, \overline{Y})$ *is a causal uniformizer for* $\varphi(\overline{P}, \overline{X}, \overline{Y})$ *over* L_1*.*

The proof of Theorem 4.13 relies on the relations between speed-independent signal languages and stutter-closed word languages developed in Section 4.3. We first establish

that each of the properties of a signal uniformizer ψ over L_1 transfers to $\psi^\dagger$ over L_2.

Lemma 4.14. *Suppose that $\psi(\overline{P}, \overline{X}, \overline{Y})$ is faithful to $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over L_1. Then $\psi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ is faithful to $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ over L_2.*

Proof. Let $\overline{\mathbf{Q}} \in L_2$, $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ be words such that $\psi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ holds over $\mathbb{N}$ and let $(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}) = S_{\mathbb{N}}(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ be corresponding signals.

Then $(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}) \in L_{\mathbb{N}}(\psi^\dagger)^\star$ and $L_{\mathbb{N}}(\psi^\dagger)^\star = L_{\mathbb{R}}(\psi)$ by Lemma 4.11, so $\psi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds over $\mathbb{R}_+$. Moreover, $\overline{\mathbf{P}} = S_{\mathbb{N}}(\overline{\mathbf{Q}})$ and $\overline{\mathbf{Q}} \in L_2 = L_1^\dagger$, so $\overline{\mathbf{P}} \in L_1^{\dagger\star} = L_1$ by Proposition 4.6 since L_1 is speed-independent. Since ψ is faithful to φ over L_1, $\varphi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds over $\mathbb{R}_+$.

By Proposition 4.9 we know that $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$ and by Lemma 4.5 we see that $W_{\mathbb{N}}(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}) = (\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$. Hence $\varphi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{Y}})$ holds over $\mathbb{N}$. Since $\overline{\mathbf{Q}}$ was an arbitrary word in L_2, $\psi^\dagger$ is faithful to φ over L_2. □

Lemma 4.15. *If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is functional over L_1 then $\psi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ is functional over L_2.*

Proof. Let $\overline{\mathbf{Q}} \in L_2$ and $\overline{\mathbf{A}}$ be words. Let $\overline{\mathbf{X}} = S_{\mathbb{N}}(\overline{\mathbf{A}})$ and $\overline{\mathbf{P}} = S_{\mathbb{N}}(\overline{\mathbf{Q}})$ be corresponding signals. It is clear that $\overline{\mathbf{P}} \in L_1$ so, since ψ is functional over L_1, there exists $\overline{\mathbf{Y}}$ such that $\psi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds over $\mathbb{R}_+$. Now let $\overline{\mathbf{B}} = W_{\mathbb{N}}(\overline{\mathbf{Y}})$. By Lemma 4.11, we know that $L_{\mathbb{N}}(\psi^\dagger)^\star = L_{\mathbb{R}}(\psi)$, so $\psi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ holds over $\mathbb{N}$.

Now let $\overline{\mathbf{C}}$ be a word such that $\psi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{C}})$ holds over $\mathbb{N}$ and let $\overline{\mathbf{Z}} = S_{\mathbb{N}}(\overline{\mathbf{C}})$ be a corresponding signal. Then $\psi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Z}})$ holds over $\mathbb{R}_+$. Since $\overline{\mathbf{P}} \in L_1$ and ψ is functional over L_1, $\overline{\mathbf{Z}} = \overline{\mathbf{Y}}$. Hence $\overline{\mathbf{C}} = \overline{\mathbf{B}}$ and $\psi^\dagger$ is functional over L_2. □

Lemma 4.16. *If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is causal over L_1 then $\psi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ is causal over L_2.*

Proof. Let $\overline{\mathbf{Q}} \in L_2$, $\overline{\mathbf{A}}$, $\overline{\mathbf{B}}$, $\overline{\mathbf{C}}$ and $\overline{\mathbf{D}}$ be words such that $\overline{\mathbf{A}}_i = \overline{\mathbf{C}}_i$ for $i \leq t$ and both $\psi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ and $\psi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{C}}, \overline{\mathbf{D}})$ hold over $\mathbb{N}$. Let $(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}) = S_{\mathbb{N}}(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ and $(\overline{\mathbf{P}}, \overline{\mathbf{U}}, \overline{\mathbf{V}}) = S_{\mathbb{N}}(\overline{\mathbf{Q}}, \overline{\mathbf{C}}, \overline{\mathbf{D}})$ be corresponding signals.

By Lemma 4.11, $L_{\mathbb{N}}(\psi^\dagger)^\star = L_{\mathbb{R}}(\psi)$ so $\psi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ and $\psi(\overline{\mathbf{P}}, \overline{\mathbf{U}}, \overline{\mathbf{V}})$ hold over $\mathbb{R}_+$. Moreover, $\overline{\mathbf{P}} = S_{\mathbb{N}}(\overline{\mathbf{Q}})$ and $\overline{\mathbf{Q}} \in L_2$, so $\overline{\mathbf{P}} \in L_1$. Since $\overline{\mathbf{X}}|_{[0,t]}$ is determined by $\overline{\mathbf{A}}_i$ for $i \leq t$, $\overline{\mathbf{U}}|_{[0,t]}$ is determined by $\overline{\mathbf{C}}_i$ for $i \leq t$ and $\overline{\mathbf{A}}_i = \overline{\mathbf{C}}_i$ for $i \leq t$, $\overline{\mathbf{X}}|_{[0,t]} = \overline{\mathbf{U}}|_{[0,t]}$. Then, since ψ is causal over L_1 and $\overline{\mathbf{X}}|_{[0,t]} = \overline{\mathbf{U}}|_{[0,t]}$, we have $\overline{\mathbf{Y}}(t) = \overline{\mathbf{V}}(t)$ by the definition of causality. Then by our definition of $\overline{\mathbf{Y}}$ and $\overline{\mathbf{V}}$, $\overline{\mathbf{B}}_t = \overline{\mathbf{D}}_t$ and thus $\psi^\dagger$ is causal over $\langle \mathbb{N}, <, \overline{\mathbf{Q}} \rangle$.

Since $\langle \mathbb{N}, <, \overline{\mathbf{Q}} \rangle$ is an arbitrary structure with $\overline{\mathbf{Q}} \in L_2$, $\psi^\dagger$ is causal over L_2. □

Having established that the properties of a signal uniformizer transfer to the equivalent word uniformizer, we now seek to reverse the direction of this implication. Note that in these cases we must insist that ψ be stutter-closed for the properties of a uniformizer to transfer to $\psi^\star$.

Lemma 4.17. *Suppose that $\psi(\overline{P}, \overline{X}, \overline{Y})$ is stutter-closed and faithful to $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ over L_2. Then $\psi^\star(\overline{P}, \overline{X}, \overline{Y})$ is faithful to $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over L_1.*

Proof. Let $\overline{\mathbf{P}} \in L_1$, $\overline{\mathbf{X}}$ and $\overline{\mathbf{Y}}$ be signals such that $\psi^\star(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds over $\mathbb{R}_+$. Let τ be a sampling sequence for $(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ and let $(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}}) = W_\tau(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ be corresponding words.

By Lemma 4.11 $L_{\mathbb{R}}(\psi^\star)^\dagger = L_{\mathbb{N}}(\psi)$ since $L_{\mathbb{N}}(\psi)$ is stutter-closed and hence $\psi(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ holds over $\mathbb{N}$. Moreover, $\overline{\mathbf{Q}} = W_\tau(\overline{\mathbf{P}})$ and $\overline{\mathbf{P}} \in L_1$, so $\overline{\mathbf{Q}} \in L_2$. Then, since ψ is faithful to $\varphi^\dagger$ over L_2, $\varphi^\dagger(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ holds over $\mathbb{N}$.

By Lemma 4.11 $L_{\mathbb{N}}(\varphi^\dagger)^\star = L_{\mathbb{R}}(\varphi)$ and by Lemma 4.5 we see that $S_\tau(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}}) = (\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$. Hence $\varphi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds over $\mathbb{R}_+$. Since $\overline{\mathbf{P}}$ was an arbitrary signal in L_1, $\psi^\star$ is faithful to φ over L_1. □

Lemma 4.18. *If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is stutter-closed and functional over L_2 then $\psi^\star(\overline{P}, \overline{X}, \overline{Y})$ is functional over L_1.*

Proof. Let $\overline{\mathbf{P}} \in L_1$ and $\overline{\mathbf{X}}$ be signals. Let τ be a sampling sequence for $(\overline{\mathbf{P}}, \overline{\mathbf{X}})$, $\overline{\mathbf{Q}} = W_\tau(\overline{\mathbf{P}})$ and $\overline{\mathbf{A}} = W_\tau(\overline{\mathbf{X}})$ be corresponding words. It is clear that $\overline{\mathbf{Q}} \in L_2$ so, since ψ is functional

over L_2, there exists $\overline{\mathbf{B}}$ such that $\psi(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ holds over $\mathbb{N}$. Now let $\overline{\mathbf{Y}} = S_\tau(\overline{\mathbf{B}})$. By Lemma 4.11, $L_{\mathbb{R}}(\psi^\star)^\dagger = L_{\mathbb{N}}(\psi)$ since $L_{\mathbb{N}}(\psi)$ is stutter-closed so $\psi^\star(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds over $\mathbb{R}_+$.

Now let $\overline{\mathbf{Z}}$ be a signal such that $\psi^\star(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Z}})$ holds over $\mathbb{R}_+$ and let τ' be a sampling sequence for $(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}, \overline{\mathbf{Z}})$. Let $(\overline{\mathbf{R}}, \overline{\mathbf{C}}, \overline{\mathbf{D}}, \overline{\mathbf{E}}) = W_{\tau'}(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}, \overline{\mathbf{Z}})$ be corresponding words. Then $\psi(\overline{\mathbf{R}}, \overline{\mathbf{C}}, \overline{\mathbf{D}})$ and $\psi(\overline{\mathbf{R}}, \overline{\mathbf{C}}, \overline{\mathbf{E}})$ both hold over $\mathbb{N}$. Again $\overline{\mathbf{R}} \in L_2$ and, since ψ is functional over L_2, $\overline{\mathbf{E}} = \overline{\mathbf{D}}$. Hence $\overline{\mathbf{Z}} = \overline{\mathbf{Y}}$ and $\psi^\star$ is functional over L_1. □

Lemma 4.19. *If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is stutter-closed and causal over L_2 then $\psi^\star(\overline{P}, \overline{X}, \overline{Y})$ is causal over L_1.*

Proof. Consider signals $\overline{\mathbf{P}} \in L_1$, $\overline{\mathbf{X}}$, $\overline{\mathbf{Y}}$, $\overline{\mathbf{U}}$ and $\overline{\mathbf{V}}$ such that $\overline{\mathbf{X}}|_{[0,t]} = \overline{\mathbf{U}}|_{[0,t]}$ and both $\psi^\star(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ and $\psi^\star(\overline{\mathbf{P}}, \overline{\mathbf{U}}, \overline{\mathbf{V}})$ hold over $\mathbb{R}_+$.

Let τ be a sampling sequence for $(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}}, \overline{\mathbf{U}}, \overline{\mathbf{V}}, \{t\})$. Then $t = \tau_{2k}$ for some k.

Let $(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}}) = W_\tau(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ and $(\overline{\mathbf{Q}}, \overline{\mathbf{C}}, \overline{\mathbf{D}}) = W_\tau(\overline{\mathbf{P}}, \overline{\mathbf{U}}, \overline{\mathbf{V}})$ be corresponding words. Since $\overline{\mathbf{A}}_i$ for $i \leq 2k$ is determined by $\overline{\mathbf{X}}|_{[0,t]}$, $\overline{\mathbf{C}}_i$ for $i \leq 2k$ is determined by $\overline{\mathbf{U}}|_{[0,t]}$ and $\overline{\mathbf{X}}|_{[0,t]} = \overline{\mathbf{U}}|_{[0,t]}$, $\overline{\mathbf{A}}_i = \overline{\mathbf{C}}_i$ for $i \leq 2k$.

We have $L_{\mathbb{R}}(\psi^\star)^\dagger = L_{\mathbb{N}}(\psi)$ by Lemma 4.11 since $L_{\mathbb{N}}(\psi)$ is stutter-closed. Therefore $\psi(\overline{\mathbf{Q}}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ and $\psi(\overline{\mathbf{Q}}, \overline{\mathbf{C}}, \overline{\mathbf{D}})$ hold over $\mathbb{N}$. Moreover, $\overline{\mathbf{Q}} = W_\tau(\overline{\mathbf{P}})$ and $\overline{\mathbf{P}} \in L_1$, so $\overline{\mathbf{Q}} \in L_2$. Then, since ψ is causal over L_2 and $\overline{\mathbf{A}}_i = \overline{\mathbf{C}}_i$ for $i \leq 2k$, we have $\overline{\mathbf{B}}_{2k} = \overline{\mathbf{D}}_{2k}$ by the definition of causality. Then by our definition of $\overline{\mathbf{B}}$ and $\overline{\mathbf{D}}$, $\overline{\mathbf{Y}}(t) = \overline{\mathbf{V}}(t)$ and thus $\psi^\star$ is causal over $\langle \mathbb{R}_+, <, \overline{\mathbf{P}} \rangle$.

Since $\langle \mathbb{R}_+, <, \overline{\mathbf{P}} \rangle$ is an arbitrary structure with $\overline{\mathbf{P}} \in L_1$, $\psi^\star$ is causal over L_1. □

We now collect these lemmas to prove Theorem 4.13.

Proof of Theorem 4.13. Let $\varphi(\overline{P}, \overline{X}, \overline{Y})$ have causal uniformizer $\psi(\overline{P}, \overline{X}, \overline{Y})$ over L_1 and consider $\psi^\dagger(\overline{P}, \overline{X}, \overline{Y})$.

By Proposition 4.9, we have that $L_{\mathbb{R}}(\psi)^\dagger = L_{\mathbb{N}}(\psi^\dagger)$. It is straightforward that $L_{\mathbb{R}}(\psi)^\dagger$ is stutter-closed and hence that $\psi^\dagger$ is stutter-closed. Moreover, by Lemmas 4.14, 4.15 and

4.16 we see that $\psi^\dagger$ satisfies the three sentences required for it to be a causal uniformizer for $\varphi^\dagger$ over L_2.

Instead let $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ have stutter-closed causal uniformizer $\psi(\overline{P}, \overline{X}, \overline{Y})$ over L_2 and consider $\psi^\star(\overline{P}, \overline{X}, \overline{Y})$.

Then by Lemmas 4.17, 4.18 and 4.19 we see that $\psi^\star$ satisfies the three sentences required for it to be a causal uniformizer for φ over L_1. □

We will use Theorem 4.13 to reduce the problem of uniformizing classes of signals, considered in Theorem 4.3, to the problem of computing stutter-closed uniformizers of stutter-closed formulae over words. We therefore undertake to prove the following Theorem.

Theorem 4.20. *Given a stutter-closed* $\mathsf{MSO}(<)$ *formula* $\varphi(\overline{P}, \overline{X}, \overline{Y})$*, one can compute a stutter-closed sentence* $\theta(\overline{P})$ *and stutter-closed formula* $\psi(\overline{P}, \overline{X}, \overline{Y})$ *such that for any stutter-closed language of words* $L \subseteq (\Sigma_P)^\omega$*,* φ *has a* $\mathsf{MSO}(<)$*-definable causal uniformizer over* L *if and only if* $L \models \theta$ *and in this case* ψ *is such a causal uniformizer.*

Considering φ over signals, we observe that the word language defined by $\varphi^\dagger$ is always stutter-closed. We first apply Theorem 4.20 then Theorem 4.13 to $\varphi^\dagger$ to derive Theorem 4.3.

4.4.3 Stutter-Closed Uniformizers

Say that a formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ defines a *stutter-preserving* relation on $\mathcal{M} = \langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$ if $\mathcal{M}$ satisfies $\forall \overline{X}, \overline{Y}(\psi(\overline{P}, \overline{X}, \overline{Y}) \rightarrow \mathit{StutPres}(\overline{P}, \overline{X}, \overline{Y}))$, where

$$\mathit{StutPres}(\overline{P}, \overline{X}, \overline{Y}) \stackrel{\text{def}}{=} \forall n\ \mathit{Odd}(n) \rightarrow \left(\left(\begin{array}{l} \overline{X}(n) = \overline{X}(n+1) = \overline{X}(n+2) \wedge \\ \overline{P}(n) = \overline{P}(n+1) = \overline{P}(n+2) \end{array} \right) \rightarrow \overline{Y}(n) = \overline{Y}(n+1) = \overline{Y}(n+2) \right) \tag{4.1}$$

and $\mathit{Odd}(n)$ defines the odd integers. In other words, in the function defined by ψ the output $\overline{Y}$ can only change when either the input $\overline{X}$ or parameters $\overline{P}$ change.

Proposition 4.21. *Let $L \subseteq (\Sigma_P)^\omega$ be a stutter-closed language of words. If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is functional over L and stutter-closed then it is also stutter-preserving over L.*

Proof. Consider any structure $\mathcal{M} = \langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$ such that $\mathcal{M} \in L$. Suppose that $\psi(\overline{P}, \overline{X}, \overline{Y})$ is stutter-closed and functional over L.

Suppose for a contradiction that we have $\overline{\mathbf{X}}$, $\overline{\mathbf{Y}}$ such that $\psi(\overline{\mathbf{P}}, \overline{\mathbf{X}}, \overline{\mathbf{Y}})$ holds but there exists an odd t such that $\overline{\mathbf{X}}(t) = \overline{\mathbf{X}}(t+1) = \overline{\mathbf{X}}(t+2)$, $\overline{\mathbf{P}}(t) = \overline{\mathbf{P}}(t+1) = \overline{\mathbf{P}}(t+2)$ and either $\overline{\mathbf{Y}}(t) \neq \overline{\mathbf{Y}}(t+1)$ or $\overline{\mathbf{Y}}(t+1) \neq \overline{\mathbf{Y}}(t+2)$.

Consider the following function, which stutters $\overline{\mathbf{X}}$ at k:

$$\delta_k(\overline{\mathbf{X}})(l) = \begin{cases} \overline{\mathbf{X}}(l) & \text{if } l < k \\ \overline{\mathbf{X}}(k) & \text{if } k \leq l \leq k+2 \\ \overline{\mathbf{X}}(l-2) & \text{if } k+2 < l \end{cases}$$

Since ψ is stutter-closed, $\psi(\delta_t(\overline{\mathbf{P}}), \delta_t(\overline{\mathbf{X}}), \delta_t(\overline{\mathbf{Y}}))$ and $\psi(\delta_{t+2}(\overline{\mathbf{P}}), \delta_{t+2}(\overline{\mathbf{X}}), \delta_{t+2}(\overline{\mathbf{Y}}))$ both hold. Note that $\delta_t(\overline{\mathbf{P}}) = \delta_{t+2}(\overline{\mathbf{P}}) \in L$ since L is stutter-closed and $\overline{\mathbf{P}}(t) = \overline{\mathbf{P}}(t+1) = \overline{\mathbf{P}}(t+2)$. Similarly, $\delta_t(\overline{\mathbf{X}}) = \delta_{t+2}(\overline{\mathbf{X}})$. However, $\delta_t(\overline{\mathbf{Y}}) \neq \delta_{t+2}(\overline{\mathbf{Y}})$ as either

$$\delta_t(\overline{\mathbf{Y}})(t+1) = \overline{\mathbf{Y}}(t) \neq \overline{\mathbf{Y}}(t+1) = \delta_{t+2}(\overline{\mathbf{Y}})(t+1) \text{ or}$$
$$\delta_t(\overline{\mathbf{Y}})(t+3) = \overline{\mathbf{Y}}(t+1) \neq \overline{\mathbf{Y}}(t+2) = \delta_{t+2}(\overline{\mathbf{Y}})(t+3)$$

This contradicts the fact that ψ is functional over L. □

Recall that we say an ω-word $u = u_0 u_1 \ldots$ is *stutter-free* if for all i either $u_{2i-1} \neq u_{2i}$ or $u_{2i} \neq u_{2i+1}$. Recall that we assume that for any structure $\langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$ one of the predicates $\mathbf{P}_i$ is not eventually constant. This means that the characteristic ω-word of the structure is stutter equivalent to a unique stutter-free word.

Proof of Theorem 4.20. We define a game based on φ such that the uniformizer ψ defines a winning strategy in this game. Our proof is based on the construction of Hänsch, Slaats

and Thomas [HST09] but requires non-trivial modification to handle various issues related to stuttering.

Step 1: *definition of game arena* $\mathcal{G}$. Define a formula $\varphi'(\overline{P}, \overline{X}, \overline{Y})$ by

$$\varphi'(\overline{P}, \overline{X}, \overline{Y}) \stackrel{\text{def}}{=} (\varphi(\overline{P}, \overline{X}, \overline{Y}) \wedge \mathit{StutPres}(\overline{P}, \overline{X}, \overline{Y})) \vee \mathit{EvConst}(\overline{P}), \tag{4.2}$$

where *StutPres* is the formula in (4.1) expressing stutter preservation and $\mathit{EvConst}(\overline{P})$ expresses that $\overline{P}$ is eventually constant. The inclusion of *StutPres* is justified by the observation in Proposition 4.21 that a stutter-closed uniformizer is stutter-preserving. The inclusion of $\mathit{EvConst}(\overline{P})$ is connected with our semantic assumption that the characteristic words of structures over $\mathbb{N}$ are not eventually constant.

The formula φ' mentions predicate names $\overline{P} = (P_1, \ldots, P_m)$ and free variables $\overline{X} = (X_1, \ldots, X_n)$ and $\overline{Y} = (Y_1, \ldots, Y_\ell)$. Then interpretations for φ' over domain $\mathbb{N}$ are ω-words over the alphabet $\{0,1\}^m \times \{0,1\}^n \times \{0,1\}^\ell$. The first step is to construct a deterministic parity automaton $\mathcal{A}$ over this alphabet that accepts precisely those words that satisfy φ'. We transform this automaton into a parity game arena $\mathcal{G}$ by separating each transition $s \xrightarrow{(p,x,y)} t$ of $\mathcal{A}$ into a pair of transitions $s \longrightarrow (s,p,x)$ and $(s,p,x) \longrightarrow t$ controlled by Player 1 and Player 2 respectively. The priorities of the vertices in $\mathcal{G}$ are inherited from $\mathcal{A}$.

Step 2: *definition of parity game* $\mathcal{G}^\pi$. Next, given a stutter-free ω-word π over alphabet $\{0,1\}^m$, representing an interpretation of the parameters, we transform the arena $\mathcal{G}$ into an infinite-state parity game denoted $\mathcal{G}^\pi$. This game is stratified into levels—one level for each letter of π, the vertices at each level being a copy of those of $\mathcal{G}$. In the odd-numbered levels, multiple rounds of the game can be played, with Player 1 controlling the passage from one level to the next. We require an odd number of rounds during each odd level, as this corresponds to stutters of the parameter word. In the even-numbered levels, only one round of the game can be played as these levels correspond to letters of the parameter word which cannot be stuttered.

The vertices of $\mathcal{G}^\pi$ are pairs consisting of a vertex of $\mathcal{G}$ and a *level number* i. For

each Player-1 edge $s \longrightarrow (s,p,x)$ in $\mathcal{G}$ and index $i \in \mathbb{N}$ we include a Player-1 edge $(s)_i \longrightarrow (s,p,x)_{i+1}$ if $p = \pi_{i+1}$. To account for the repetition allowed in odd levels, we also add edges $(s)_{2i+1} \longrightarrow (\hat{s},p,x)_{2i+1}$ and $(\hat{s})_{2i+1} \longrightarrow (s,p,x)_{2i+1}$ if $p = \pi_{2i+1}$. For each Player-2 edge $(s,p,x) \longrightarrow t$ in $\mathcal{G}$ and index $i \in \mathbb{N}$ we include edges $(s,p,x)_i \longrightarrow (t)_i$ and $(\hat{s},p,x)_{2i+1} \longrightarrow (\hat{t})_{2i+1}$ in $\mathcal{G}^\pi$. This use of $\hat{s}$ vertices in odd levels ensures that an odd number of rounds are played as required. Finally we add a new initial Player-1 vertex $(\hat{s})_0$ to $\mathcal{G}^\pi$, where s is the initial vertex of $\mathcal{G}$. From this vertex there is an edge to a vertex $(s,p,x)_0$ if $p = \pi_0$.

Note that due to the disjunct $EvConst(\overline{P})$ in (4.2), Player 1 cannot win $\mathcal{G}^\pi$ by choosing never to leave a given level.

The key property of the game $\mathcal{G}^\pi$, which depends on the fact that $L(\mathcal{A})$ is stutter-closed, is as follows:

> *Player 2 wins $\mathcal{G}^\pi$ if and only if $\varphi(\overline{P},\overline{X},\overline{Y})$ has a causal uniformizer over the class of all infinite words π' that are stutter equivalent to π.*

The easier direction in the proof of the above claim is the right-to-left implication: one can easily show that a causal uniformizer for $\varphi(\overline{P},\overline{X},\overline{Y})$ over the stutter equivalence class of π yields a winning strategy in $\mathcal{G}^\pi$. Below we concentrate on the left-to-right implication.

Step 3: *coding and testing strategies.* As $\mathcal{G}^\pi$ is a parity game it is determined and has memoryless winning strategies. To compute winning strategies we first divide the set of game states into levels S_i which contain those nodes annotated with level number i. We can encode the possible levels by a finite alphabet Σ and thus represent the game as an ω-word $\sigma \in \Sigma^\omega$ in which σ_i represents level S_i. Note that σ can be produced as the output of a transducer $\mathcal{S}$ on input π.

A memoryless strategy for Player 2 in $\mathcal{G}^\pi$ maps each vertex $(s,p,x)_i$ to a vertex $(t)_i$ (and each $(\hat{s},p,x)_i$ to a vertex $(\hat{t})_i$) and can be represented by a word γ over a finite alphabet Γ whose letters encode the finite sub-strategy for each level i. We build a deterministic parity

automaton $\mathcal{T}$ that takes as input pairs of words π and γ and accepts if and only if the strategy γ is winning in $\mathcal{G}^\pi$. The automaton $\mathcal{T}$ incorporates the transducer $\mathcal{S}$ to transform the input word π into a word σ representing the game $\mathcal{G}^\pi$ in the manner described above. For each level i of $\mathcal{G}^\pi$ and level-i Player-2 strategy γ_i, automaton $\mathcal{T}$ computes the finite set of all possibilities over Player-1 moves for the first state, last state and lowest-priority intermediate state of the level-i segment of a play of $\mathcal{G}^\pi$.

The strategy tester automaton $\mathcal{T}$ is equivalent to an $\mathsf{MSO}(<)$ formula $\chi(\overline{P}, \overline{S})$, where $\overline{S}$ encodes letters of the strategy alphabet Γ. Note that χ is only satisfied by stutter-free interpretations of $\overline{P}$.

Step 4: *selecting a winning strategy.* The formula $\chi(\overline{P}, \overline{S})$ allows us to detect when $\overline{S}$ encodes a winning strategy in $\mathcal{G}^\pi$. A key issue here is that there may be more than one winning strategy for a given set of parameters π: a uniformizer corresponds to a particular winning strategy.

We can compute an $\mathsf{MSO}(<)$ formula that picks a winning strategy using a result of Lifsches and Shelah [LS98] on the computability of selectors in $\mathsf{MSO}(<)$. We say that a formula $\alpha(\overline{P}, \overline{S})$ is a *selector* for a formula $\beta(\overline{P}, \overline{S})$ over a structure $\mathcal{M} = \langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$ if:

1. $\mathcal{M} \models \exists^{\leq 1} \overline{S}\, \alpha(\overline{P}, \overline{S})$;
2. $\mathcal{M} \models \forall \overline{S} (\alpha(\overline{P}, \overline{S}) \rightarrow \beta(\overline{P}, \overline{S}))$;
3. $\mathcal{M} \models (\exists \overline{S} \beta(\overline{P}, \overline{S})) \rightarrow (\exists \overline{S} \alpha(\overline{P}, \overline{S}))$.

Lemma 4.22 (Selector Lemma [Rab07b])**.** *There is an algorithm that for every formula $\beta(\overline{P}, \overline{S})$ constructs a formula $\alpha(\overline{P}, \overline{S})$ such that α is a selector for β over all structures $\mathcal{M} = \langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$.*

Applying Lemma 4.22 we can compute a selector $\chi'(\overline{P}, \overline{S})$ for $\chi(\overline{P}, \overline{S})$. Then $\chi'(\overline{P}, \overline{S})$ is satisfied when $\overline{P}$ is interpreted by a stutter-free word π and $\overline{S}$ is interpreted by a word representing a winning strategy γ in $\mathcal{G}^\pi$.

Step 5: *definition of* $\theta(\overline{P})$ *and* $\psi(\overline{P}, \overline{X}, \overline{Y})$.

Similar to the definition of the strategy tester automaton we can compute a formula $\mathit{Strat}(\overline{P}, \overline{X}, \overline{Y}, \overline{S})$ that is true precisely when in $\mathcal{G}^\pi$, for π the unique stutter-free word equivalent to $\overline{P}$, the sequence of moves $\overline{X}$ by Player 1 generates the sequence of responses $\overline{Y}$ by Player 2 given that his strategy on the k-th round is $\overline{S}(k)$.

Note that $\mathit{Strat}(\overline{P}, \overline{X}, \overline{Y}, \overline{S})$ defines a stutter-closed language. Closure under removing stutters follows from the fact that $\overline{S}$ encodes a memoryless strategy and (it can be assumed without loss of generality that) $\mathcal{A}$ doesn't change state when its input stutters. Closure under adding stutters follows similarly using in addition the fact that $\overline{S}$ encodes a stutter-preserving strategy.

We also define $\chi''(\overline{P}, \overline{S})$ to be the stutter-closure of the strategy selection operator $\chi'(\overline{P}, \overline{S})$.

Finally we are able to define

$$\begin{aligned} \theta(\overline{P}) &\stackrel{\text{def}}{=} \exists \overline{S} \chi''(\overline{P}, \overline{S}) \\ \psi(\overline{P}, \overline{X}, \overline{Y}) &\stackrel{\text{def}}{=} \exists \overline{S} \left(\chi''(\overline{P}, \overline{S}) \wedge \mathit{Strat}(\overline{P}, \overline{X}, \overline{Y}, \overline{S}) \right) . \end{aligned}$$

Then both θ and ψ are stutter-closed since both χ'' and *Strat* are stutter-closed.

By construction, $\theta(\overline{\mathbf{P}})$ holds if and only if there exists $\overline{\mathbf{S}}$ that encodes a winning strategy for Player 2 in $\mathcal{G}^\pi$, where π is the unique stutter-free word equivalent to the characteristic ω-word $u_{\overline{\mathbf{P}}}$. Since *Strat* encodes plays of this game that follow this winning strategy, ψ uniformizes φ.

This concludes the proof of Theorem 4.20. □

4.5 Uniformizing Metric Formulae

In this section we show decidability of the uniformization problem for $\mathsf{MSO}(<, +1)$ over bounded real time domains.

Note that as an immediate corollary of Theorem 4.3, we can establish an analogous result for $\mathsf{MSO}(<)$ over bounded domains.

Corollary 4.23. *Let $\mathbb{T} = [0, N)$ be a bounded interval of reals. Given an $\mathsf{MSO}(<)$ formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$, one can compute a sentence $\theta(\overline{P})$ and formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ such that for every structure $\mathcal{M} = \langle \mathbb{T}, <, \overline{\mathbf{P}} \rangle$, φ has a causal uniformizer over $\mathcal{M}$ if and only if $\mathcal{M} \models \theta$ and in this case ψ is such a causal uniformizer.*

We now seek to apply this result to formulae of $\mathsf{MSO}(<, +1)$ by first removing all references to the $+1$ relation using the translation of Section 3.5.2. This translation provides a bijection between formulae φ over structures $\langle [0, N), <, +1, \mathbf{P} \rangle$ and formulae $\overline{\varphi}$ over structures $\langle [0, 1), <, \mathbf{P}_0, \ldots, \mathbf{P}_{N-1} \rangle$ without the $+1$ relation where each predicate X over $[0, N)$ is replaced by N predicates $X_0, \ldots, X_N$ over $[0, 1)$.

4.5.1 Main Result

The following result, concerning the computability of uniformizers in $\mathsf{MSO}(<, +1)$, is the main result of this chapter. Since considering labelled intervals is essential in the proof and the technique generalises straightforwardly, we prove a more general result involving uniformization of $\mathsf{MSO}(<, +1)$ over labelled intervals than we stated in the introduction.

Theorem 4.4. *Let $\mathbb{T} = [0, N)$ be a bounded interval of reals. Given an $\mathsf{MSO}(<, +1)$ formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$, one can compute a sentence $\theta(\overline{P})$ and formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ such that for every structure $\mathcal{M} = \langle \mathbb{T}, <, +1, \overline{\mathbf{P}} \rangle$, φ has an $\mathsf{MSO}(<, +1)$-definable causal uniformizer over $\mathcal{M}$ if and only if $\mathcal{M} \models \theta$ and in this case ψ is such a causal uniformizer over $\mathcal{M}$.*

Proof. To simplify notation, we consider the special case where φ has only X and Y as free variables as well as referring to a single structure label P.

Step 1. Applying the transformation described in Section 3.5.2 to $\varphi(P, X, Y)$ yields an $\mathsf{MSO}(<)$ formula $\overline{\varphi}(P_0, \ldots, P_{N-1}, X_0, \ldots, X_{N-1}, Y_0, \ldots, Y_{N-1})$, such that there is a natural

bijection between the models of φ over $[0, N)$ and the models of $\overline{\varphi}$ over $[0, 1)$, where $X_i(t)$ holds if and only if $X(i + t)$ holds for $i = 0, 1, \ldots, N - 1$ and $0 \leq t < 1$.

Step 2. We reduce the problem of uniformizing φ over $\langle \mathbb{T}, <, +1, \mathbf{P} \rangle$ to an *N-phase* uniformization procedure applied to $\overline{\varphi}$. In the first phase, we construct a causal operator to determine the values of Y_0 from the values of X_0. The second phase then constructs a causal operator to determine the values of Y_1 from those of X_1, treating the values of X_0 and Y_0 generated in the previous phase as parameters that are already fixed. At the end of the N-th phase we wish $\overline{\varphi}$ to be satisfied by the values of $\overline{X}$ and $\overline{Y}$ we have determined as well as the set of parameters $\overline{P}$. Our claim is that we can construct a series of functions in such a scenario if and only if we can uniformize φ over $\langle \mathbb{T}, <, +1, \mathbf{P} \rangle$.

We formalise the above idea by defining N uniformization problems $G_0, G_1 \ldots, G_{N-1}$ involving only $\mathsf{MSO}(<)$ over $[0, 1)$. The basic data of each problem G_k, $0 \leq k < N$ are illustrated in Figure 4.1.

We define the problems G_k by backward induction, starting with G_{N-1}. The winning condition in this problem is

$$\varphi_{N-1}(P_0, \ldots, P_{N-1}, X_0, \ldots, X_{N-1}, Y_0, \ldots, Y_{N-1}) \stackrel{\text{def}}{=}$$
$$\overline{\varphi}(P_0, \ldots, P_{N-1}, X_0, \ldots, X_{N-1}, Y_0, \ldots, Y_{N-1}),$$

where $X_0, \ldots, X_{N-2}$ and $Y_0, \ldots, Y_{N-2}$ are considered as parameters additional to the original parameters $P_0, \ldots, P_{N-1}$ and X_{N-1} and Y_{N-1} are considered as variables. Applying Corollary 4.23 we obtain a domain formula θ_{N-1} and uniformizer ψ_{N-1} for φ_{N-1}.

Suppose that we have defined G_k, with basic data as given in Figure 4.1. Then we define G_{k-1} as follows. The formula to be uniformized, denoted φ_{k-1}, is defined to be the domain formula θ_k from the preceding problem G_k. In G_{k-1} we consider $X_0, \ldots, X_{k-2}$ and $Y_0, \ldots, Y_{k-2}$ as parameters additional to $P_0, \ldots, P_{N-1}$ and X_{k-1} and Y_{k-1} as variables. The domain formula θ_{k-1} and uniformizer ψ_{k-1} are then obtained by applying Corollary 4.23 to φ_{k-1}.

Uniformization Problem G_k

Input:

Winning condition	$\varphi_k(P_0, \ldots, P_{N-1}, X_0, \ldots, X_k, Y_0, \ldots, Y_k)$
Parameters	$P_0, \ldots, P_{N-1}, X_0, \ldots, X_{k-1}, Y_0, \ldots, Y_{k-1}$
Variables	X_k, Y_k

Output:

Domain formula	$\theta_k(P_0, \ldots, P_{N-1}, X_0, \ldots, X_{k-1}, Y_0, \ldots, Y_{k-1})$
Uniformizer	$\psi_k(P_0, \ldots, P_{N-1}, X_0, \ldots, X_k, Y_0, \ldots, Y_k)$

Figure 4.1: Basic data for G_k.

Step 3: *Definition of domain formula $\theta(P)$ and uniformizer $\psi(P, X, Y)$ for $\varphi(P, X, Y)$.* Let $\overline{\psi} \stackrel{\text{def}}{=} \psi_0 \wedge \cdots \wedge \psi_{N-1}$ and $\overline{\theta} \stackrel{\text{def}}{=} \theta_0$, then $\overline{\psi}$ is a formula in variables $X_0, \ldots, X_{N-1}$, $Y_0, \ldots, Y_{N-1}$ and parameters $P_0, \ldots, P_{N-1}$ whilst $\overline{\theta}$ is a sentence in parameters $P_0, \ldots, P_{N-1}$. The $\mathsf{MSO}(<, +1)$ formula $\psi(P, X, Y)$ and sentence $\theta(P)$ are obtained from $\overline{\psi}$ and $\overline{\theta}$ respectively by replacing everywhere $P_i(t)$ with $P(i + t)$, $X_i(t)$ with $X(i + t)$ and $Y_i(t)$ with $Y(i + t)$. The transformation from $\overline{\psi}$ and $\overline{\theta}$ to ψ and θ can be seen as the reverse of the transformation in Section 3.5.2, motivating our choice of notation.

This completes the description of the procedure to decide whether $\varphi(P, X, Y)$ has a uniformizer and if so to construct such a uniformizer. We turn now to the correctness of this construction.

Fix a structure $\mathcal{M} = \langle \mathbb{T}, <, +1, \mathbf{P} \rangle$ and suppose that $\mathcal{M} \models \theta$. We show that $\psi(P, X, Y)$ defines a causal uniformizer for $\varphi(X, Y)$ on $\mathcal{M}$. Let $\overline{\mathcal{M}} = \langle [0, 1), <, \mathbf{P}_0, \ldots, \mathbf{P}_{N-1} \rangle$.

Let $\mathbf{X} \subseteq \mathbb{T}$. We must show that there is a unique $\mathbf{Y} \subseteq \mathbb{T}$ such that $\psi(\mathbf{P}, \mathbf{X}, \mathbf{Y})$ and for this $\mathbf{Y}$ also $\varphi(\mathbf{P}, \mathbf{X}, \mathbf{Y})$. Write $\overline{\mathbf{X}} = \mathbf{X}_0, \ldots, \mathbf{X}_{N-1}$ for the tuple of subsets of $[0, 1)$ defined by $\mathbf{X}_i(t)$ if and only if $\mathbf{X}(i + t)$.

Since $\mathcal{M} \models \theta$, we see by Proposition 3.8 that $\overline{\mathcal{M}} \models \overline{\theta}$ and hence that ψ_0 uniformizes φ_0 over $\overline{\mathcal{M}}$. Thus there exists $\mathbf{Y}_0 \subseteq [0,1)$ such that $\psi_0(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{Y}_0)$ and $\varphi_0(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{Y}_0)$ both hold. But $\theta_1 \equiv \varphi_0$, so $\theta_1(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{Y}_0)$ also holds over $[0,1)$. Since ψ_1 uniformizes φ_1 over this structure, there exists $\mathbf{Y}_1 \subseteq [0,1)$ such that $\psi_1(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1)$ and $\varphi_1(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1)$ both hold. Continuing in this vein we generate successively predicates $\mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1}$ such that

$$\psi_0(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{Y}_0) \wedge \psi_1(\overline{\mathbf{P}}, \mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1) \wedge \cdots \wedge \psi_{N-1}(\overline{\mathbf{P}}, \mathbf{X}_0, \ldots, \mathbf{X}_{N-1}, \mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1})$$

holds over $[0,1)$. Thus by definition of $\overline{\psi}$ we have $\overline{\psi}(\overline{\mathbf{P}}, \mathbf{X}_0, \ldots, \mathbf{X}_{N-1}, \mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1})$.

Now define $\mathbf{Y} \subseteq \mathbb{T}$ by having $\mathbf{Y}(t+i)$ hold if and only if $\mathbf{Y}_i(t)$ holds for $i = 0, \ldots, N-1$ and $0 \leq t < 1$. Then by definition of ψ we have $\psi(\mathbf{P}, \mathbf{X}, \mathbf{Y})$. Furthermore, since ψ_{N-1} uniformizes φ_{N-1} we also have that

$$\varphi_{N-1}(\overline{\mathbf{P}}, \mathbf{X}_0, \ldots, \mathbf{X}_{N-1}, \mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1})$$

holds over $[0,1)$. But φ_{N-1} was defined to be $\overline{\varphi}$ and thus by Proposition 3.8 we have that $\varphi(\mathbf{P}, \mathbf{X}, \mathbf{Y})$ also holds over $\mathbb{T}$.

The fact that ψ is functional and causal can easily be obtained from the corresponding properties of $\psi_0, \ldots, \psi_{N-1}$ in the above construction. Reversing the above argument also allows us to deduce that if $\varphi(P, X, Y)$ has a uniformizer over $\mathcal{M}$ then $\mathcal{M} \models \theta$: given a uniformizer $\psi(P, X, Y)$ for φ one successively generates uniformizers for φ_{N-1} down to φ_0.□

4.6 Lower Bounds

Recall the family of functions $\exp_k : \mathbb{N} \to \mathbb{N}$ from Section 3.6.2 where $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$. Recall that a function $f : \mathbb{N} \to \mathbb{N}$ is *non-elementary* if it grows faster than any $\exp_k$.

The procedure for uniformizing $\mathsf{MSO}(<,+1)$ formulae over bounded time domain $\mathbb{T} = [0, N)$ described in Section 4.5.1 has non-elementary complexity. This blow-up arises not only from the non-elementary transformation of $\mathsf{MSO}(<,+1)$ to automata — repeated application of Corollary 4.23 leads to an N-fold exponential blow-up.

In this section we give a non-elementary lower bound for the bounded uniformization problem for $\mathsf{FO}(<,+1)$ that holds even for formulae of a fixed quantifier alternation depth (for which satisfiability over bounded intervals is elementary[1]). This is proven by reduction from the language emptiness problem for star-free regular expressions. The construction we outline below can also be used to show that uniformization is non-elementary also for the temporal logic MTL for which satisfiability over bounded intervals is ExpSpace-complete [ORW09].

4.6.1 Star-Free Regular Expressions

A *star-free regular expression* over alphabet Σ is built from the symbols $\emptyset$ and σ, for any $\sigma \in \Sigma$, using the operations of union $(+)$, concatenation $(\cdot)$, and complementation $(\neg)$. Such an expression E denotes a language $L(E) \subseteq \Sigma^*$ which is defined as follows:

$$L(\emptyset) = \emptyset \text{ and } L(\sigma) = \{\sigma\};$$

$$L(\neg E) = \Sigma^* \setminus L(E);$$

$$L(E + E') = L(E) \cup L(E');$$

$$L(E \cdot E') = L(E) \cdot L(E').$$

The following result was shown in [SM73].

Theorem 4.24. *The language emptiness problem for star-free regular expressions is non-elementary.*

[1] The paper [ORW09] provides an elementary reduction of the satisfiability problem for $\mathsf{FO}(<,+1)$ over bounded intervals to the problem for $\mathsf{FO}(<)$ which does not change the quantifier alternation depth.

The *operator depth* $\mathrm{odp}(E)$ of a star-free regular expression E is defined as follows:

$$\begin{aligned}
\mathrm{odp}(\emptyset) &= \mathrm{odp}(\sigma) = 1; \\
\mathrm{odp}(\neg E) &= \mathrm{odp}(E); \\
\mathrm{odp}(E + E') &= \max\{\mathrm{odp}(E), \mathrm{odp}(E')\} + 1; \\
\mathrm{odp}(E \cdot E') &= \max\{\mathrm{odp}(E), \mathrm{odp}(E')\} + 1.
\end{aligned}$$

Note that negation does not count toward the operator depth.

Given a star-free regular expression E over alphabet Σ and a word $w = w_0 w_1 \ldots w_{n-1} \in \Sigma^*$ we define the *membership game* $\mathbb{G}(w, E)$. This is a two-player game with N rounds, where N is the operator depth of E. The two players are *Prover*, who is trying to show $w \in E$, and *Refuter*, who is trying to show $w \notin E$. The positions of the game are triples (b, e, F) where b and e are positions in the word w and F has the form G or $\neg G$ for G a sub-expression of E. The initial position is $(0, n, E)$. If the position at the start of a given round is (b, e, F) the goal of Prover is to show that $w_b w_{b+1} \ldots w_{e-1} \in F$. The round proceeds as follows:

- If $F \equiv F_1 \cdot F_2$ then Prover moves first by choosing an index i with $b \leq i \leq e$. Refuter responds by selecting either (b, i, F_1) or (i, e, F_2) as the position in the next round;

- If $F \equiv \neg(F_1 \cdot F_2)$ then Refuter moves first by choosing an index i with $b \leq i \leq e$. Prover responds by selecting either $(b, i, \neg F_1)$ or $(i, e, \neg F_2)$ as the position in the next round;

- If $F \equiv F_1 + F_2$ then Prover selects either (b, e, F_1) or (b, e, F_2) as the position in the next round;

- If $F \equiv \neg(F_1 + F_2)$ then Refuter selects either $(b, e, \neg F_1)$ or $(b, e, \neg F_2)$ as the position in the next round.

The positions (b, e, σ), $(b, e, \neg\sigma)$, $(b, e, \emptyset)$ and $(b, e, \neg\emptyset)$ are terminal and are classified as winning for Prover or Refuter according to whether $w_b, w_{b+1} \ldots, w_{e-1}$ is a member of the corresponding expression.

It is clear that Prover has a winning strategy in $\mathbb{G}(w, E)$ if and only if $w \in L(E)$.

4.6.2 Encoding in $\mathsf{FO}(<, +1)$

For any regular expression E, let $Sub(E)$ be the set of sub-expressions of E along with their negations (we identify $\neg\neg E$ with E). For example, if $E = \sigma_1 + (\sigma_2 \cdot \neg\emptyset)$,

$$Sub(E) = \{\emptyset, \neg\emptyset, \sigma_1, \neg\sigma_1, \sigma_2, \neg\sigma_2, (\sigma_2 \cdot \neg\emptyset), \neg(\sigma_2 \cdot \neg\emptyset), (\sigma_1 + (\sigma_2 \cdot \neg\emptyset)), \neg(\sigma_1 + (\sigma_2 \cdot \neg\emptyset))\}\,.$$

Given that a position in $\mathbb{G}(w, E)$ is a triple from the set $\Pi \stackrel{\text{def}}{=} \{0, \ldots, n\}^2 \times Sub(E)$, a *play* of $\mathbb{G}(w, E)$ can be represented as a word in Π^* denoting a sequence of successive positions. The idea of our reduction is to encode plays as signals over a domain $[0, N+1)$ and to construct a formula of $\mathsf{FO}(<, +1)$ that is satisfied by a signal if and only it encodes a winning play for Prover. In this encoding successive game positions are encoded in successive unit-length subintervals of the domain.

Our encoding represents plays using the following set of monadic predicates. For each element $F \in Sub(E)$, we have predicates R_F, P_F and G_F. For each $\sigma \in \Sigma$, we have a predicate σ; by a slight abuse of notation, we refer to this subset of predicates as Σ also. We take predicates b and e to represent the indices used in $\mathbb{G}(w, E)$ and have three final predicates: R_{split}, P_{split} and $\#$. We use $\Sigma_\#$ to refer to $\Sigma \cup \{\#\}$. We divide these predicates into two tuples $\overline{X} = \{R_F \mid F \in Sub(E)\} \cup \{R_{split}\}$ and $\overline{Y}$ which contains all the other predicates. We intend for the predicates in $\overline{X}$ to be controlled by the Refuter whilst those in $\overline{Y}$ are controlled by the Prover. An example of how such a signal will correspond to successive rounds of $\mathbb{G}(w, E)$ is shown in Figure 4.2.

Let E be a star-free regular expression of operator depth N and write $\mathbb{T} = [0, N)$. We

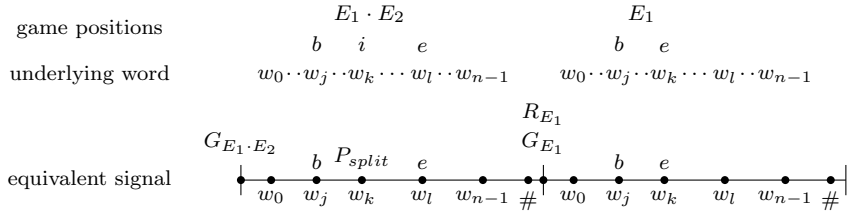


Figure 4.2: A signal encoding a play

construct a formula (of quantifier alternation depth three) $\varphi_E(\overline{X}, \overline{Y})$ such that φ_E has a uniformizer over $\langle \mathbb{T}, <, +1 \rangle$ if and only if $L(E) \neq \emptyset$.

This formula is structured as $\varphi_E \equiv \varphi_{prules} \wedge (\neg\varphi_{rrules} \vee (\varphi_{win}))$. Intuitively this means that the formula is false if Prover breaks the rules, otherwise true if Refuter breaks the rules or if the input represents a play of the game which is won by Prover. We describe the construction of these subformulae as a series of conjunctive clauses.

First, we wish Prover to encode the word he wishes to show is a member of this regular expression in the first time unit and insist that he copies this forward precisely to each other time unit. We do this by adding the following rules to φ_{prules}

$$\bigwedge_{\sigma \in \Sigma_\#} \left[\forall t \left(\sigma(t) \to \bigwedge_{\gamma \in \Sigma_\# \setminus \{\sigma\}} \neg\gamma(t) \right) \wedge \forall t \forall u \left(t + 1 = u \to (\sigma(t) \leftrightarrow \sigma(u)) \right) \right]$$

and note that since Prover may choose to demonstrate that the empty word is a member of this regular expression, this formula does not require him to place any letters in the first time unit. However, we always insist he places a number of markers in the first time unit. We require that $\#$ is placed after the end of the word, b on the first letter and e on the hash symbol by adding

$$\exists t \left[t < 1 \wedge \#(t) \wedge e(t) \wedge \forall u \left(t < u < 1 \to \bigwedge_{\sigma \in \Sigma_\#} \neg\sigma(u) \right) \right]$$

and

$$\exists t \left[t < 1 \wedge b(t) \wedge \bigvee_{\sigma \in \Sigma_\#} \sigma(t) \wedge \forall u \left(0 < u < t \to \bigwedge_{\gamma \in \Sigma_\#} \neg\gamma(u) \right) \right]$$

to φ_{prules}. The special case where he wishes to represent the empty word is handled by placing both b and e on the hash symbol.

We use the predicates G_F, R_F and P_F for $F \in Sub(E)$ to represent the Prover's obligation to show that the subword between the markers b and e is contained in the subexpression F. The P and R predicates represent this obligation being generated by a move of the Prover and Refuter respectively, with the G predicates required to hold whenever either the corresponding P or R predicate holds. Hence for every element $F \in Sub(E)$ we add $\forall t(P_F(t) \vee R_F(t)) \rightarrow G_F(t)$ to φ_{prules}. Moreover, at the start of the game, the Prover has an obligation to show that the entire word belongs to E, hence we add $G_E(0)$ to φ_{prules}.

Next, we come to the rules which correspond to the nonterminal rounds of the game. These rounds induce rules for the behaviour of both Refuter and Prover and also determine where the markers b and e are copied to for the next round of the game. Note that in all these formulae we use the shorthand $b(t+1)$ and so on because the obligation on the left side of the implications should not hold after time $N-1$, as each round corresponds to one time unit and each reduces the nesting depth of the sub-expression considered.

For each element of $Sub(E)$ of the form $(E_1 + E_2)$, we add

$$\forall t \left[G_{(E_1+E_2)}(t) \rightarrow \left(\begin{array}{c} \exists u \, (t < u < t+1 \wedge b(u) \wedge b(u+1)) \\ \wedge \exists u \, (t < u < t+1 \wedge e(u) \wedge e(u+1)) \\ \wedge ((P_{E_1}(t+1) \wedge \neg P_{E_2}(t+1)) \vee (P_{E_2}(t+1) \wedge \neg P_{E_1}(t+1))) \end{array} \right) \right]$$

to φ_{prules}. Here the Prover chooses whether the next obligation should be E_1 or E_2 and updates the b and e delimiters to point to the same letters in the next round.

For each element of $Sub(E)$ of the form $\neg(E_1 + E_2)$, we add

$$\forall t \left[G_{\neg(E_1+E_2)}(t) \rightarrow \left(\begin{array}{c} \exists u (t < u < t+1 \wedge b(u) \wedge b(u+1)) \\ \wedge \exists u (t < u < t+1 \wedge e(u) \wedge e(u+1)) \end{array} \right) \right]$$

to φ_{prules} and

$$\forall t \left[G_{\neg(E_1+E_2)}(t) \rightarrow ((R_{\neg E_1}(t+1) \wedge \neg R_{\neg E_2}(t+1)) \vee (R_{\neg E_2}(t+1) \wedge \neg R_{\neg E_1}(t+1))) \right]$$

to φ_{rrules}. Here the Prover updates the b and e delimiters to point to the same letters in the next round while the Refuter chooses whether the next obligation should be $\neg E_1$ or $\neg E_2$.

For each element of $Sub(E)$ of the form $(E_1 \cdot E_2)$, we add

$$\forall t \left[G_{(E_1 \cdot E_2)}(t) \rightarrow \left(\begin{array}{l} G_{E_1}(t+1) \rightarrow \left(\begin{array}{c} \exists u\,(t < u < t+1 \wedge b(u) \wedge b(u+1)) \\ \wedge \exists u\,(t < u < t+1 \wedge P_{split}(u) \wedge e(u+1)) \end{array} \right) \\ \wedge G_{E_2}(t+1) \rightarrow \left(\begin{array}{c} \exists u\,(t < u < t+1 \wedge P_{split}(u) \wedge b(u+1)) \\ \wedge \exists u\,(t < u < t+1 \wedge e(u) \wedge e(u+1)) \end{array} \right) \\ \wedge \exists u\,(t < u < t+1 \wedge P_{split}(u)) \end{array} \right) \right]$$

to φ_{prules} and

$$\forall t \left[G_{(E_1 \cdot E_2)}(t) \rightarrow ((R_{E_1}(t+1) \wedge \neg R_{E_2}(t+1)) \vee (R_{E_2}(t+1) \wedge \neg R_{E_1}(t+1))) \right]$$

to φ_{rrules}. Here, the Prover splits the word into two subwords with the proposition P_{split}, the Refuter chooses whether the next obligation should be E_1 with the first subword or E_2 with the second subword and the Prover updates the b and e delimiters correspondingly.

For each element of $Sub(E)$ of the form $\neg(E_1 \cdot E_2)$, we add

$$\forall t \left[G_{\neg(E_1 \cdot E_2)}(t) \rightarrow \left(\begin{array}{l} G_{\neg E_1}(t+1) \rightarrow \left(\begin{array}{c} \exists u\,(t < u < t+1 \wedge b(u) \wedge b(u+1)) \\ \wedge \exists u\,(t < u < t+1 \wedge R_{split}(u) \wedge e(u+1)) \end{array} \right) \\ \wedge G_{\neg E_2}(t+1) \rightarrow \left(\begin{array}{c} \exists u\,(t < u < t+1 \wedge R_{split}(u) \wedge b(u+1)) \\ \wedge \exists u\,(t < u < t+1 \wedge e(u) \wedge e(u+1)) \end{array} \right) \\ \wedge ((P_{\neg E_1}(t+1) \wedge \neg P_{\neg E_2}(t+1)) \vee (P_{\neg E_2}(t+1) \wedge \neg P_{\neg E_1}(t+1))) \end{array} \right) \right]$$

to φ_{prules} and

$$\forall t \left[G_{\neg(E_1 \cdot E_2)}(t) \rightarrow \exists u\,(t < u < t+1 \wedge R_{split}(u)) \right]$$

to φ_{rrules}. Here, the Refuter splits the word into two subwords with the proposition R_{split}, then the Prover chooses whether the next obligation should be $\neg E_1$ with the first subword or $\neg E_2$ with the second subword and updates the b and e delimiters correspondingly.

To prevent Refuter from introducing spurious R_F obligations, we analyse what obligations are able to generate them according to the rules of the game and introduce rules to ensure that these obligations are the only legal antecedents. For each element $F \in Sub(E)$, let

$$C_F = \begin{array}{l} \{(F \cdot E_2) \in Sub(E) \mid E_2 \in Sub(E)\} \cup \{(E_1 \cdot F) \in Sub(E) \mid E_1 \in Sub(E)\} \\ \cup \{(F + E_2) \in Sub(E) \mid E_2 \in Sub(E)\} \cup \{(E_1 + F) \in Sub(E) \mid E_1 \in Sub(E)\} \end{array}.$$

Then add the following to φ_{rrules} for each $F \in Sub(E)$:

$$\forall t \left[R_F(t+1) \to \bigvee_{H \in C_F} G_H(t) \right]$$

Finally, we describe the rules of φ_{win}, which correspond to terminal rounds in the game.

For each element of $Sub(E)$ of the form σ, we add

$$\forall t \left[G_\sigma(t) \to \exists u \left(\begin{array}{c} t < u < t+1 \wedge b(u) \wedge \sigma(u) \\ \wedge \exists v \left(\begin{array}{l} u < v < t+1 \wedge e(v) \wedge \bigvee_{\gamma \in \Sigma_\#} \gamma(v) \\ \wedge \forall w \left(u < w < v \to \bigwedge_{\gamma \in \Sigma_\#} \neg\gamma(w) \right) \end{array} \right) \end{array} \right) \right]$$

to φ_{win}. In order for the game to be won by the Prover, we require that the single letter σ is contained between the b and e delimiters.

For each element of $Sub(E)$ of the form $\neg\sigma$, we add

$$\forall t \left[G_{\neg\sigma}(t) \to \exists u \left(\begin{array}{c} t < u < t+1 \wedge b(u) \wedge \bigvee_{\gamma \in \Sigma_\#} \gamma(u) \\ \wedge \left(\neg\sigma(u) \vee \exists v \left(\begin{array}{l} u < v < t+1 \wedge e(v) \wedge \bigvee_{\gamma \in \Sigma_\#} \gamma(v) \\ \wedge \exists w \left(u < w < v \to \bigvee_{\gamma \in \Sigma} \gamma(w) \right) \end{array} \right) \right) \end{array} \right) \right]$$

to φ_{win}. In this case, the game is won by the Prover if the subword contained between the b and e delimiters is not σ. This happens if either it does not begin with σ or has more than one letter.

If $\emptyset \in Sub(E)$, we add $\forall t \left[G_\emptyset(t) \to \exists u \left(t < u < t+1 \wedge b(u) \wedge e(u) \right) \right]$ to φ_{win}. Here, the game is won by Prover if the b and e delimiters mark the same letter, denoting the empty word.

If $\neg\emptyset \in Sub(E)$, we add

$$\forall t \left[G_{\neg\emptyset}(t) \rightarrow \exists u \left(\begin{array}{c} t < u < t+1 \wedge b(u) \wedge \bigvee_{\sigma\in\Sigma} \sigma(u) \\ \wedge \exists v \left(u < v < t+1 \wedge e(v) \wedge \bigvee_{\gamma\in\Sigma_\#} \gamma(v) \right) \end{array} \right) \right]$$

to φ_{win}. Finally, in this case the game is won by the Prover if the b and e delimiters mark different letters, denoting a non-empty word.

This completes the definition of φ_E.

We now turn our attention to the complexity of this construction. Note that the size of $Sub(E)$ is linear in the size of E and that the number of subformulae generated by the above construction is linear in the size of $Sub(E)$. Further note that the length of each subformula is either linear in the size of E, linear in the size of Σ or constant regardless of E or Σ. Hence the size of φ_E is linear in the sizes of E and Σ and can be constructed from E and Σ in logarithmic space by maintaining a constant number of pointers to positions of E or Σ. We therefore conclude the following proposition:

Proposition 4.25. *Given an alphabet Σ and star-free regular expression E of operator depth N, one can compute a formula (of quantifier alternation depth three) $\varphi_E(\overline{X}, \overline{Y})$ of $\mathsf{FO}(<, +1)$ whose size is linear in $|E|$ and $|\Sigma|$ such that φ_E has a causal uniformizer over $\langle [0, N), <, +1\rangle$ if and only if $L(E) \neq \emptyset$. Moreover, this computation can be completed in space logarithmic in $|E|$ and $|\Sigma|$.*

Combining this proposition with Theorem 4.24, we immediately obtain

Theorem 4.26. *The time-bounded uniformization problem for $\mathsf{FO}(<, +1)$ is non-elementary for formulae of quantifier alternation depth at most three.*

4.7 Summary

In this chapter, we solved a number of parametric extensions of Church's synthesis problem where we require the uniformizer to be definable in the same monadic second-order logic

as the specification. Whilst these parameters are not our primary object of study, they are crucial to our proofs.

Over the classical domain of the natural numbers, we extended previously known decidability results to the case where both the specification and uniformizer are stutter-closed.

We then considered the continuous-time domain of the reals under the finite-variability interpretation. When only the $<$ relation is available, we proved decidability of the synthesis problem over unbounded intervals by reduction to the stutter-closed case over the natural numbers. When the logic is augmented with the $+1$ relation, we proved decidability of the synthesis problem over bounded intervals of reals by using a series of uniformization problems to reduce to the case of $\mathsf{MSO}(<)$ over $[0, 1)$.

We finally derived a non-elementary lower bound for the synthesis problem for formulae of $\mathsf{FO}(<, +1)$ of quantifier alternation depth three; the satisfiability problem is, by contrast, elementary for such formulae.

Chapter 5

Safety Alternating Timed Automata

5.1 Introduction

In this chapter, we investigate the language emptiness problem for alternating timed automata over infinite words. We recall that one obtains the complement of an alternating timed automaton simply by exchanging $\wedge$ and $\vee$ in transitions and complementing the set of accepting states. The consequence of this is that the language emptiness problem for alternating timed automata subsumes the universality problem for timed automata.

It was shown in [AD94] that the universality problem for timed automata is undecidable over infinite words if the automaton has at least two clocks and in [ADOW05] this result was refined to show that even with one clock the problem is undecidable in general.

These negative results led to a search for a more restricted class of alternating timed automata whose language emptiness problem is decidable over infinite words. Ouaknine and Worrell identified such a class in [OW06b] whose members have three restrictions: only a single clock is used, all states are accepting (known as a *safety* property) and the clock is reset every time the state changes (known as a *locality* property). Their proof of decidability first translated to a region automaton before employing a version of Higman's

lemma. Ouaknine and Worrell also showed that it is possible to translate formulae from the *Safety* fragment of Metric Temporal Logic into these local 1-clock safety alternating timed automata and thereby decide the satisfiability problem for this logic.

We extend these results by introducing the auxiliary formalism of *channel machines*. These machines are a variant of Turing machines with a channel which acts as a queue to which items can be added and removed instead of the tape which allows for symbols to be replaced in-place. Since they are Turing-powerful [BZ83], we need to consider some restrictions on their expressiveness in order to restore decidability. One possible restriction is the idea of bounding the number of times the machine can *cycle* its channel (i.e. read the current contents while possibly adding new symbols); this can be seen as analogous to bounding the number of head-reversals of a Turing machine and the problem of cycle-bounded reachability was explored in [BMOW07].

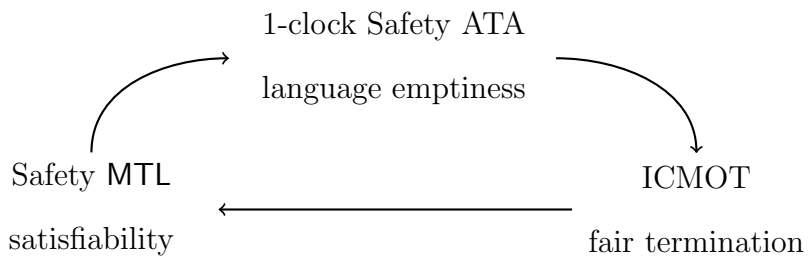
Another type of restriction is to introduce *errors* into the channel. Machines with *lossiness errors* (where some symbols are written but never subsequently read) were first defined in [AJ96] and were employed by Lasota and Walukiewicz to show the non-primitive recursive hardness of language emptiness for 1-clock alternating timed automata over finite words [LW05]. Ouaknine and Worrell used machines with *insertion errors* (where some symbols are read but were not previously written), first defined in [CFI96], in their proof of the same result [OW05]. The two models have been shown to have different complexities when infinite computations are considered, however; termination of insertion channel machines was shown to be primitive recursive in [BMO+08] (even with additional *occurrence testing* operations), but termination of lossy channel machines was shown by Schnoebelen to be non-primitive recursive [Sch02] and in fact not even multiply recursive [CS08].

In the first part of this chapter, we focus on the termination problem for insertion channel machines but add a *fairness* requirement which corresponds to the non-Zeno requirement for words consumed by alternating timed automata. In Section 5.3, we inves-

tigate this *Fair Termination* problem for insertion channel machines without additional operations and show that it is PSPACE-complete. We then consider the Fair Termination problem when occurrence tests are added and show in Section 5.4 that the problem is decidable by using a *well-quasi-ordering* on configurations of the machine.

Occurrence tests greatly increase the complexity of insertion channel machines and we recall the first few levels of the *extended Grzegorczyk hierarchy* of fast-growing functions in order to describe it [Grz53, LW70, CS08]. This hierarchy is defined by induction, with $F_0(n) \stackrel{\text{def}}{=} n+1$ and $F_{i+1}(n) \stackrel{\text{def}}{=} F_i^{n+1}(n)$, so $F_1(n) = 2n+1$, $F_2(n) = \Theta(n^2)$, $F_3(n) = \Theta\left(\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n}\right)$ and the Ackermann function is $F_\omega(n) \stackrel{\text{def}}{=} F_n(n)$. Without our fairness requirement, the termination problem for insertion channel machines with occurrence tests is already non-elementary, sitting at level F_3 of the hierarchy. Our first main result is then that the fair termination problem is harder still — in Section 5.5 we show that the problem is hard for $\text{SPACE}(F_4(n))$, but we do not exclude the possibility of a primitive recursive decision procedure.

In the second part of this chapter, we show that the fair termination problem for insertion channel machines with occurrence tests (ICMOTs) is interreducible with the satisfiability problem for Safety MTL formulae and the language emptiness problem for 1-clock safety alternating timed automata by exhibiting reductions as depicted in the diagram below. The main consequence of this interreducibility is that the language emptiness problem for 1-clock safety alternating timed automata is decidable but $\text{SPACE}(F_4(n))$ hard.



The reduction from Safety MTL formulae to 1-clock safety alternating timed automata

recounted in Section 5.7.1 was first proved in [OW06b], but the other two reductions we present are novel. In Section 5.6.2 we illustrate how the fair termination problem for an insertion channel machine with renaming can be encoded as a Safety MTL formula and in Section 5.7.2 we exhibit a reduction from 1-clock safety alternating timed automata to insertion channel machines with occurrence testing.

We note that this approach to extending the results of [OW06b] stands in contrast to the later work of Parys and Walukiewicz, who identified in [PW09] the class of *weak alternating timed automata* (which have a single clock and no transitions from accepting to non-accepting states) and gave a decision procedure through a direct construction.

5.2 Insertion Channel Machines

A *channel machine* is a variant of a Turing machine. Instead of a tape which it can read from and write to, the machine operates with a channel which is a queue; a FIFO data structure where a read operation can only consume the symbol at the head of the channel and a write symbol must place the symbol at the end of the channel. There is no *a priori* bound on the size of the channel, so channel machines are infinite-state systems.

During the action of a Turing machine we observe that the head can move at most one place (either to the left or the right) with each operation, so we can determine the contents of a tape cell in a new configuration by reading its current content and the content of the cells to its left and right. This means we can update a Turing machine configuration stored on a channel within the FIFO constraint, buffering the letter from one cell as part of the channel machine's state and writing the corresponding new configuration as we go. Since there is no bound on the size of the channel, we can simulate an unrestricted Turing machine in this way, so channel machines are also undecidable systems [BZ83].

5.2.1 Types of Channel Machine

Starting with a finite channel alphabet Σ, we obtain $Op = \{\sigma!, \sigma? \mid \sigma \in \Sigma\} \cup \{nop\}$, $Op_Z = Op \cup \{zero(\sigma) \mid \sigma \in \Sigma\}$ and $Op_R = Op \cup \{R \subseteq \Sigma \times (\Sigma \cup \{\varepsilon\})\}$, the possible sets of operations on that alphabet.

Intuitively, $\sigma!$ is a write, $\sigma?$ is a read and $zero(\sigma)$ ensures that there are no σ symbols on the channel. *nop* denotes an internal transition of the machine which does not change the channel while R denotes a renaming transition, whose meaning depends on what each letter is related to. If σ is related to exactly one letter μ, all instances of σ on the channel are changed to μ; if σ is related to ε, instances of σ on the channel are instead deleted; if σ is related to two or more letters, each instance is nondeterministically renamed (so the channel may contain a mixture of the letters σ was related to). If σ is not related to any letters, this transition behaves the same as $zero(\sigma)$ in addition to its effect on the other letters, hence we can consider $zero(\sigma)$ as the renaming $\{(\alpha, \alpha) \mid \alpha \in (\Sigma \setminus \{\sigma\})\}$.

Definition 5.1. A *channel machine with renaming* (CMR) is a tuple $\mathcal{C} = (S, s_0, \Sigma, \Delta)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Σ is the finite channel alphabet and $\Delta \subseteq S \times Op_R \times S$ is a set of transition rules. Note that Δ need not be a function, so channel machines allow nondeterminism.

A *channel machine with occurrence testing* (CMOT) replaces Op_R with Op_Z, while we use *channel machine* (CM) to refer to a machine which uses only operations from Op.

It is clear that every CM can be considered as a CMOT, and by our observation that occurrence tests can be implemented as special renamings, it is further clear that every CMOT can be considered as a CMR.

A *configuration* of $\mathcal{C}$ is a pair $\gamma = (s, x)$ where $s \in S$ is the state and $x = x_1 \dots x_n \in \Sigma^*$ is the channel contents. If we define

$$R(x) \stackrel{\text{def}}{=} \{y_1 \dots y_n \in \Sigma^* \mid x_i R y_i, i = 1, \dots, n\} ,$$

the rules in Δ induce an (error-free) transition relation on the set of configurations according to the following table:

Rule	Transition
$(s, \sigma!, t)$	$(s, x) \xrightarrow{\sigma!} (t, x \cdot \sigma)$
$(s, \sigma?, t)$	$(s, \sigma \cdot x) \xrightarrow{\sigma?} (t, x)$
$(s, zero(\sigma), t)$	$(s, x) \xrightarrow{zero(\sigma)} (t, x)$, if $\sigma \notin x$
(s, R, t)	$(s, x) \xrightarrow{R} (t, y)$, if $y \in R(x)$
(s, nop, t)	$(s, x) \xrightarrow{nop} (t, x)$

We say that o is the *operand* of a transition of the form $(s, x) \xrightarrow{o} (t, y)$. We define an (error-free) run of $\mathcal{C}$ with input $w \in \Sigma^*$ to be a (finite or infinite) sequence of transitions and configurations of the form $\gamma_0 \xrightarrow{o_0} \gamma_1 \xrightarrow{o_1} \gamma_2 \xrightarrow{o_2} \cdots$ which follows the relation defined above and has $\gamma_0 = (s_0, w)$.

5.2.2 Machines with Errors

Since this model of channel machines is Turing powerful, we must reduce its expressive power in order to recover decidability. Instead of a more typical restriction in resources (e.g. the size of the channel or the number of computation steps), we introduce *errors* which increase the nondeterminism in the computation of the machine.

The type of errors we consider are known as *insertion* or *read* errors. The intuition behind this model is that the channel may gain extra symbols erroneously and we adopt a "lazy" semantics to represent them. This semantics merely extends the transition relation induced by Δ with one new type of rule. If $(s, \sigma?, t)$ is in Δ, a transition $(s, x) \xrightarrow{\sigma?} (t, x)$ is induced (for any channel contents x) that does not affect the contents of the channel. This corresponds to the symbol σ being an insertion error; we do not record its presence on the channel in any other way, preferring to assume that it was inserted just in time to be read.

We refer to a machine with this semantics as an *insertion channel machine* (possibly with renaming or occurrence testing).

5.2.3 Fairness

This model of insertion channel machines has been studied in the literature [CFI96, OW05, OW06a, BMO+08] and has restored decidability for a number of problems. Without renaming or occurrence testing operations, it was noted in [CFI96] that all transitions are constantly enabled so most verification problems for these machines reduce to the associated problems for finite automata.

Intuitively, one of the problems with this model is that every read may be an error, so we cannot be sure that anything written to the channel is ever read. In [BMO+08], insertion channel machines are permitted multiple channels to remedy this deficiency as well as being allowed occurrence tests and it is shown that for insertion channel machines with occurrence testing and k channels, the termination problem has k-ExpSpace complexity. In contrast to this approach, we introduce the notion of a *fair* run in which every symbol written to the channel is eventually read.

Definition 5.2. A *fair run* of $\mathcal{C}$ with input $w \in \Sigma^*$ is a (finite or infinite) sequence of transitions and configurations of the form $\gamma_0 \xrightarrow{o_0} \gamma_1 \xrightarrow{o_1} \gamma_2 \xrightarrow{o_2} \cdots$, with $\gamma_0 = (s_0, w)$, for which there exists a strictly monotone partial function $f : \mathbb{N} \to \mathbb{N}$ such that for each $o_i = \sigma!$, there exists $j = f(i)$ with $(s_j, y \cdot \sigma) \xrightarrow{\sigma?} (s_{j+1}, y)$.

The function f in this definition maps the step at which the symbol σ is written to the channel to the step at which it is read, hence we require it to be strictly monotone as each symbol written must be read in turn.

We can also consider *cycles* of the channel — if the channel currently contains letters $\alpha_1, \ldots, \alpha_n$ we say that the channel has been cycled when α_n has been read and removed

from the channel (as opposed to read by an error transition). If a fair run has an infinite number of write transitions, then it must have an infinite number of cycles (as every symbol written must be read); conversely if a run has an infinite number of cycles, then it must be fair. Thus in this sense, the notions of fairness and cycles are equivalent, at least for runs with infinitely many write transitions.

For the rest of this section, we are concerned with the following problem:

Definition 5.3 (Fair Termination Problem)**.** Given an insertion channel machine (ICM, ICMOT or ICMR) $\mathcal{C}$, decide whether all fair runs of $\mathcal{C}$ with empty initial channel contents are finite.

Working with the entire set of runs of $\mathcal{C}$ may prove problematic, so we investigate the converse of this question: does there exist an infinite fair run of $\mathcal{C}$ with empty input?

5.3 Fair Termination for ICMs

In this section we investigate the fair termination problem for insertion channel machines which do not have renaming or occurrence testing operations.

5.3.1 PSPACE Hardness

Recall that a linear bounded Turing machine has a tape which contains no cells other than those occupied by the letters of the input. We recall that the acceptance problem for such a machine is PSPACE-complete and seek to reduce this problem to the fair termination problem for insertion channel machines.

Proposition 5.4. *For every (nondeterministic) linear bounded Turing machine $\mathcal{A}$ and input w, we can construct an insertion channel machine $\mathcal{C}(\mathcal{A}, w)$ of size quadratic in $|\mathcal{A}|$ and $|w|$ such that $\mathcal{C}(\mathcal{A}, w)$ has an infinite fair run on empty input if and only if $\mathcal{A}$ accepts w. Moreover, this construction can be performed in space logarithmic in $|\mathcal{A}|$ and $|w|$.*

Proof. The idea of this reduction is for the channel machine $\mathcal{C}(\mathcal{A}, w)$ to guess an entire accepting computation history of $\mathcal{A}$ on w, place it on the channel, and then verify it. While the channel machine suffers from insertion errors, we can compute what the correct length of the channel should be from the size of w, then use this result to detect any insertion errors. We divide these tasks so that the machine searches for insertion errors only after guessing and verifying an accepting computation history of $\mathcal{A}$ on w, then enters an infinite loop if no errors were detected (halting if either an insertion error or an error with the guessed computation history is detected).

If $\mathcal{A} = (S, s_0, \Sigma, \Delta, F)$, we define the channel alphabet of $\mathcal{C}(\mathcal{A}, w)$ to be

$$[((\Sigma \cup (\Sigma \times S)) \times \{\cdot, \bot\}) \cup \{\#\}] \times \{+, -\} .$$

The idea is that we record the position of the Turing machine's head with the current state and the progress of our verification with a $\cdot$. We also have a # symbol at the start of each configuration as a divider (see Figure 5.1 for details). Each symbol has a positive and negative copy to allow us to track the progress of the machine - at each stage we will either read positive and write negative symbols or *vice versa*. This distinction enables us to detect when we have cycled the channel, even if the contents is very long.

$\#^+$	$(s_0, w_1)^+$	w_2^+	$\cdots$	w_n^+	$\#^+$	x^+	$(s_1, w_2)^+$	$\cdots$	w_n^+	$\cdots$	$\#^+$	y_1^+	y_2^+	$\cdots$	$(s_j, y_i)^+$	$\cdots$	y_n^+

Figure 5.1: A computation history as stored on the channel

From the initial state of $\mathcal{C}(\mathcal{A}, w)$, we allow the machine to write any positive symbols it wishes, but not read anything. This phase allows the machine to nondeterministically guess an accepting computation history of $\mathcal{A}$ on w, but cannot continue infinitely in a fair run since no symbols are read. The machine can move onto the next phase at any time.

The next phase of the computation of $\mathcal{C}(\mathcal{A}, w)$ is to verify its guess of the computation history of $\mathcal{A}$ on w. We intend this phase to be length preserving, that is, in a perfect

run the length of the channel will not change during this phase (whenever a letter is read, another one is written).

This phase works by first placing a dot above each symbol which follows a # (we presume these to be the initial tape cells of configurations of $\mathcal{A}$), then on each subsequent cycle verifying that the symbols under the dots form a correct succession before advancing the dots by one symbol. This stage also ensures that the initial state of $\mathcal{A}$ is s_0, its initial tape contents is w and its final state is in F. We require $\mathcal{C}(\mathcal{A}, w)$ to have a number of states proportional to $|w|$ to implement this phase of the computation (and hard-wire the ith cycle of the phase to look for the dot i symbols after each #) in order to control the number of cycles and hence the length of the tape contents in each guessed configuration of $\mathcal{A}$ (recall that we can detect cycles of the channel by using the polarity of the symbols).

If our channel machine has a perfect run which reaches this point, we can be sure that $\mathcal{A}$ accepts w; the machine halts immediately if any of the testing cycles detects an error with its guess of the computation history of $\mathcal{A}$. We then use the next phase of computation to ensure that any insertion errors which have occurred are detected. We adopt the following reasoning: each configuration of $\mathcal{A}$ has $|w|$ cells plus a # divider and there should be at most $2^{|w|}$ such configurations; without loss of generality, we assume that every accepting run of $\mathcal{A}$ on w uses exactly $2^{|w|}$ steps. Hence the channel should have exactly $(|w|+1)2^{|w|}$ symbols during the entire testing phase above (as moving dots and changing polarity does not alter the number of symbols). Since any insertion errors will increase the length of the channel, they may be detected by halving the size of the channel $|w|$ times, which would leave exactly $|w|+1$ symbols remaining on the channel if there were no insertion errors.

Thus the error-detection consists of $|w|$ cycles which each attempt to halve the length of the channel by entering a loop which reads any two symbols of one polarity and writes one of the opposite polarity back. While the precise symbols are unimportant at this point (and so could all be replaced by #), their polarity still allows us to ensure that this phase

cycles the channel exactly $|w|$ times while halving. At the end of this phase, we have a cycle which ensures that the channel has precisely $|w|+1$ symbols. The machine will again immediately halt if it detects an insertion error, so we know that any run which continues past this phase must have been perfect until now.

Finally, the channel machine enters a single state from which it loops by performing *nop* operations. Since every other loop writes symbols which cannot be read on that loop, every infinite fair run of $\mathcal{C}(\mathcal{A}, w)$ must reach this state and hence must correspond to correctly guessing an accepting computation history of $\mathcal{A}$ on w. Moreover such a guess will lead to this state via a perfect run. Hence $\mathcal{C}(\mathcal{A}, w)$ has an infinite fair insertion run with initially empty channel if and only if $\mathcal{A}$ accepts w.

Note that the verification phase of $\mathcal{C}(\mathcal{A}, w)$ requires a number of cycles equal to $|w|$ and that each such cycle requires a number of states linear in $|w|$. Moreover each channel halving cycle of computation needs fewer states than the cycles in the verification phase, hence $\mathcal{C}(\mathcal{A}, w)$ needs a number of states quadratic in $|w|$ in total. Further observe that the alphabet of $\mathcal{C}(\mathcal{A}, w)$ may be at most quadratic in $|\mathcal{A}|$ since it has a symbol for each combination of an element from S and an element from Σ. Each state in $\mathcal{C}(\mathcal{A}, w)$ may therefore require a number of outgoing transitions quadratic in $|\mathcal{A}|$, hence the size of $\mathcal{C}(\mathcal{A}, w)$ will be $O(|\mathcal{A}|^2|w|^2)$.

In order to see that this construction may be performed in space logarithmic in $|\mathcal{A}|$ and $|w|$, one observes that only a constant number of pointers to elements of $\mathcal{A}$ or w need to be used to construct each section of $\mathcal{C}(\mathcal{A}, w)$. □

Due to this proposition and the fact that a linear bounded Turing machine has a PSPACE-hard acceptance problem, we conclude that deciding whether an insertion channel machine has an infinite fair run from an empty input is PSPACE-hard.

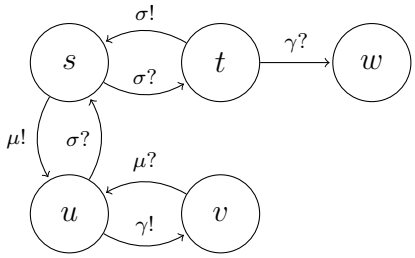


Figure 5.2: Example of $\mathcal{C}$

5.3.2 Decidability in PSPACE

In this section we show that if an insertion channel machine has a sufficiently long fair run, it must have an infinite fair run. In this case "sufficiently long" will refer to the number of times the run cycles the channel. We state the main technical result of this section as follows.

Proposition 5.5. *For every insertion channel machine $\mathcal{C} = (S, s_0, \Sigma, \Delta)$ (which does not use renaming or occurrence testing), if there exists a run which cycles the channel more than $6|S|$ times then there exists an infinite fair run.*

In order to prove this proposition, we construct a function which assigns a rank to each configuration of the machine and then show that this rank is non-increasing during any run of the machine. We then observe that a run of the machine that cycles the channel whilst remaining in the same rank can be extended to an infinite fair run in a straightforward way. We illustrate the construction of the ranking function using the channel machine $\mathcal{C}$ depicted in Figure 5.2.

Defining a ranking

Consider the labelled graph induced by (S, Δ), where vertices are members of S and edges are induced by elements of Δ, labelled by elements of Op. For example if $(s, \sigma!, t) \in \Delta$

there is an edge from s to t in the graph labelled by $\sigma!$. We construct an ordered tree $T_{\mathcal{C}}$ from the graph in a top down fashion in two stages.

In the first stage, we label the root node by the entire graph (S, Δ) and if (S, Δ) is not strongly connected, we identify its strongly connected components (SCCs) and use these to label the children of the root in an order determined as follows. We topologically sort the set of strongly connected components as $\{(S_i, \Delta_i) \mid i \leq n\}$ for some n such that:

- $\bigcup_{i=1}^{n} S_i = S$,
- $S_i \cap S_j = \emptyset$ for $i \neq j$,
- if $(s, op, t) \in \Delta$ with $s \in S_i$ and $t \in S_j$ then $i \leq j$, and
- if $(s, op, t) \in \Delta$ with $s, t \in S_i$ then $(s, op, t) \in \Delta_i$.

We label the ith child of the root by (S_i, Δ_i) for $i = 1, \ldots, n$. This stage is illustrated in Figure 5.3.

For any SCC (Q, Γ), let the letters written by transitions in (Q, Γ) be $\mathit{Writes}(Q, \Gamma)$ and the letters read by transitions in (Q, Γ) be $\mathit{Reads}(Q, \Gamma)$. We call an SCC (Q, Γ) *safe* if $\mathit{Writes}(Q, \Gamma) \subseteq \mathit{Reads}(Q, \Gamma)$.

In the second stage of the construction of our ordered tree $T_{\mathcal{C}}$ we employ a recursive procedure, repeating the following process for each leaf node labelled by an unsafe SCC; we observe that after our initial stage each leaf node will be an SCC.

Let the currently considered leaf be labelled by (Q, Γ) and let

$$\Gamma' = \Gamma \setminus \{(s, \sigma!, t) \mid s, t \in Q, \sigma \in \mathit{Writes}(Q, \Gamma) \setminus \mathit{Reads}(Q, \Gamma)\}.$$

Next, identify the strongly connected components of the subgraph induced by (Q, Γ'); intuitively, this subgraph corresponds to deleting those transitions which write a letter not read in Q. As in the first stage, we add a child node for each SCC and order these children by the topological ordering in (Q, Γ'). Note that we allow SCCs which contain only one

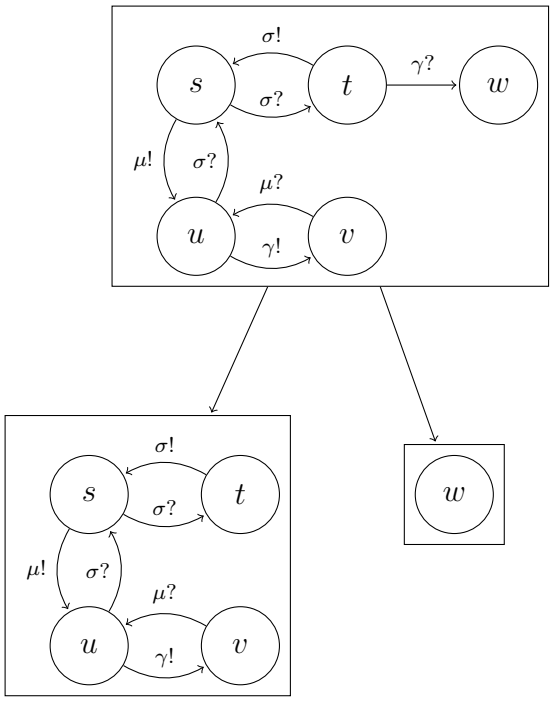


Figure 5.3: First stage of construction of $T_{\mathcal{C}}$

state so the set $\{(Q_i, \Gamma_i) \mid i \leq n\}$ of SCCs forms a partition of (Q, Γ'). We add n children to the node labelled by (Q, Γ) and label the ith child of by (Q_i, Γ_i), for $i = 1, \ldots, n$. The result of this process is illustrated in Figure 5.4.

There are two cases following the application of this step. Either there is a single child node, which now will be safe (and therefore a leaf) because of the construction of Γ', or there are at least 2 child nodes. With at least 2 child nodes, each has strictly fewer vertices than its parent. We hence see that this process must terminate because at each step we either reduce the number of vertices in the node being considered or immediately produce a leaf.

Since the leaf nodes form a partition of the vertices in the initial graph, there must be at most $|S|$ of them. Thus we infer from the construction process that the tree has at most $3|S|$ nodes in total. To each node, we associate two ranks, one "alive" (A) and the other "dead" (D). Ranks are ordered as follows: each parent has lower ranks than its children

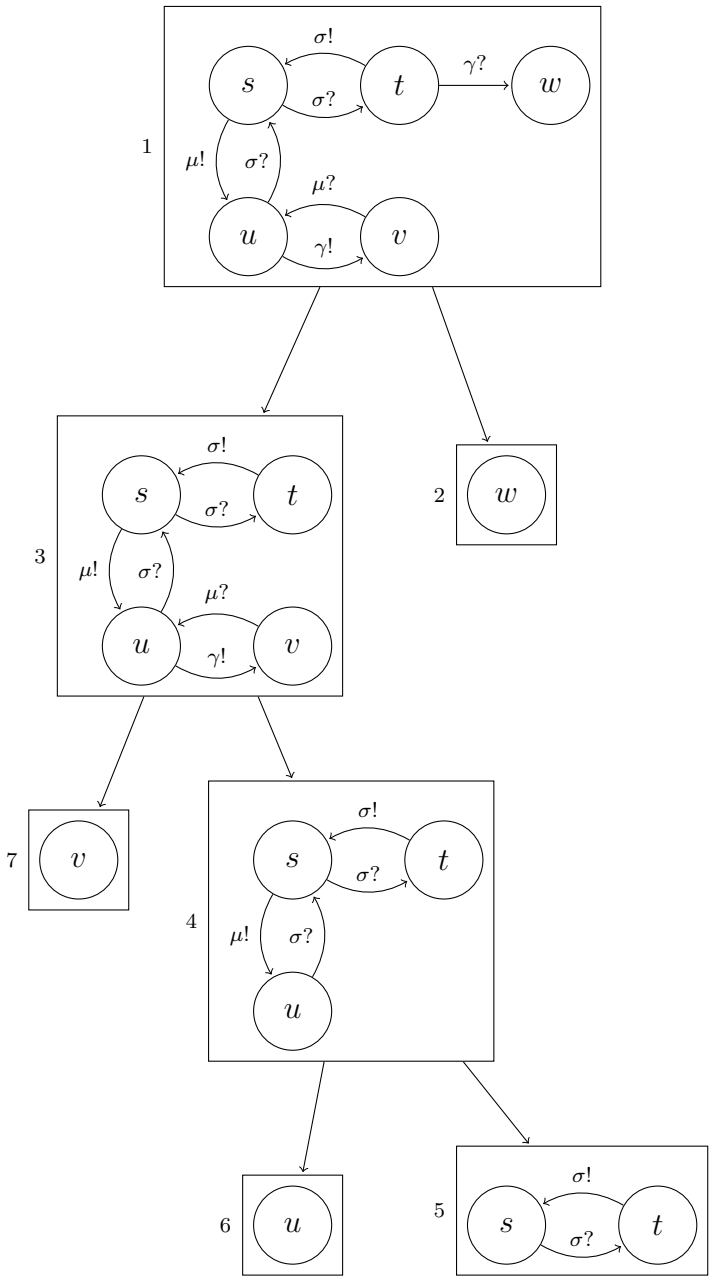


Figure 5.4: Second stage of construction of $T_{\mathcal{C}}$ with ranking order

and all ranks in the subtree rooted at the ith child of a node are higher than those in the subtree rooted at the jth child if $i < j$. Within the ranks for a particular node, the "alive" rank is higher than the "dead" one. (Thus the highest ranked nodes are at the bottom left of the tree.) This ranking is illustrated in Figure 5.4.

We now describe how the rank of each configuration of a run is assigned. Initially, the run starts in state s_0 with an empty channel and is assigned rank $(\mathcal{N}, A)$ where $\mathcal{N}$ is the leaf node of the tree containing s_0. Following this, we assign a rank to the configuration $\gamma = (s, \alpha_1\alpha_2 \dots \alpha_k)$ by the following recursive procedure which starts at the root of the tree $T_{\mathcal{C}}$:

When considering a node labelled by (Q, Γ), there are three cases:

- If $\alpha_i \notin \mathit{Reads}(Q, \Gamma)$ for some α_i, set $\mathit{rank}(\gamma) = ((Q, \Gamma), D)$.
- If $\alpha_i \in \mathit{Reads}(Q, \Gamma)$ for all $1 \leq i \leq k$ and this node is a leaf, set $\mathit{rank}(\gamma) = ((Q, \Gamma), A)$.
- If $\alpha_i \in \mathit{Reads}(Q, \Gamma)$ for all $1 \leq i \leq k$ and this node is not a leaf, consider next the child labelled by (Q_j, Γ_j) where $s \in Q_j$ (which exists by construction).

The intuition here is that the configuration will be either be assigned an A-rank associated to a leaf node (if the contents of the channel can be read without leaving the SCC at that leaf) or the D-rank associated with the highest node whose SCC must be left in order to read the current channel contents. For example, the configuration $(s, \sigma\sigma\sigma)$ would be assigned rank $(5, A)$ in the example $T_{\mathcal{C}}$ of Figure 5.4 while $(u, \mu\sigma)$ would be assigned rank $(4, D)$ as $\mu\sigma$ can be read by transitions in both the SCCs at node 1 and at node 3, but not by the SCC at node 4.

Note that $\mathit{rank}(s_0, \emptyset) = (N, A)$ where N is the label of the leaf node of the tree $T_{\mathcal{C}}$ containing s_0. We now state three properties of this ranking which can easily be observed.

Lemma 5.6. *1. If $\mathit{rank}(s, \alpha) = ((Q, \Gamma), \mu)$ for $\mu \in \{A, D\}$ then $s \in Q$.*

2. *If* $rank(s, \alpha_1\alpha_2 \dots \alpha_k) = ((Q, \Gamma), A)$, *then* $\alpha_i \in Reads(Q, \Gamma)$ *for all* $1 \leq i \leq k$.

3. *If* $rank(s, \alpha_1\alpha_2 \dots \alpha_k) = ((Q, \Gamma), D)$, *then there exists* $1 \leq i \leq k$ *such that* $\alpha_i \notin Reads(Q, \Gamma)$.

Ranks are non-increasing during a run

Now that we have defined a ranking on the configurations of $\mathcal{C}$, we show that the ranks we assign to successive configurations of a run are non-increasing.

Lemma 5.7. *For any run* $\gamma_0 \xrightarrow{o_0} \gamma_1 \xrightarrow{o_1} \gamma_2 \xrightarrow{o_2} \cdots$ *of* $\mathcal{C}$ *and* $i < j$, $rank(\gamma_i) \geq rank(\gamma_j)$.

Proof. It is clear that since the ranks form a transitive order we need only show that $rank(\gamma_i) \geq rank(\gamma_{i+1})$ for any pair of successive configurations γ_i and γ_{i+1}.

Consider a configuration (s, α) with rank associated with node (Q, Γ) of the tree $T_{\mathcal{C}}$ and suppose that after one transition the configuration becomes (t, β) where $\alpha = \alpha_1 \dots \alpha_k$ and $\beta = \beta_1 \dots \beta_l$.

First consider the case where the rank of (t, β) is associated with the same node (Q, Γ). Since $((Q, \Gamma), A) > ((Q, \Gamma), D)$, the only nontrivial case is $rank(s, \alpha) = ((Q, \Gamma), D)$. In this case, the third property of Lemma 5.6 means that there must be some letter α_i which cannot be read by a transition in Γ. However, $s, t \in Q$ and Γ contains all read transitions between vertices in Q. Therefore, α_i must still be contained in the word β and (t, β) must have rank $((Q, \Gamma), D)$.

Otherwise, the rank of (t, β) is associated with some node (R, Λ) for $(Q, \Gamma) \neq (R, \Lambda)$.

Suppose that (Q, Γ) is an ancestor of (R, Λ). Then (Q, Γ) is not a leaf, so the process for assigning ranks to configurations can only have assigned (s, α) the rank $((Q, \Gamma), D)$. Then by the third property of Lemma 5.6, there must be some α_i with $\alpha_i \notin Reads(Q, \Gamma)$. As (Q, Γ) is an ancestor of (R, Λ), the transition from (s, α) to (t, β) must be contained in Γ and therefore α_i cannot have been read so $\alpha_i = \beta_j$ for some j. Thus (t, β) must have

rank $((Q, \Gamma), D)$, which contradicts the assumption that $(Q, \Gamma) \neq (R, \Lambda)$, so this case is impossible.

If instead (R, Λ) is an ancestor of (Q, Γ), any rank associated with (Q, Γ) is higher than one associated with its ancestor (R, Λ), so the rank has decreased between (s, α) and (t, β).

Finally, suppose that (V, Υ) is the lowest common ancestor of (Q, Γ) and (R, Λ) (with $(Q, \Gamma) \neq (V, \Upsilon) \neq (R, \Lambda)$). Further suppose that s is contained in the ith child of (V, Υ) and t in the jth child.

The case $i = j$ is not possible since (V, Υ) is the lowest common ancestor of (Q, Γ) and (R, Λ).

The case $i > j$ is not possible since the only transitions which could have moved from s to t are ones in $\mathit{Writes}(V, \Upsilon) \setminus \mathit{Reads}(V, \Upsilon)$ due to the reachability ordering of (V, Υ)'s children. Thus the rank of (t, β) must be associated with (V, Υ) or an ancestor, which contradicts the fact that the rank of (t, β) is associated with the node (R, Λ) which is a descendant of (V, Υ).

Hence the only case is $i < j$ and then the rank associated with (R, Λ) is lower than that associated with (Q, Γ), since (R, Λ) is included in the subtree rooted at the jth child of (V, Υ).

This completes the proof that in all cases the rank is non-increasing. □

Long runs imply infinite ones

Finally, we show that the existence of a sufficiently long run implies the existence of an infinite fair run and repeat the statement of the main result of this section:

Proposition 5.5. *For every insertion channel machine $\mathcal{C} = (S, s_0, \Sigma, \Delta)$ (which does not use renaming or occurrence testing), if there exists a run which cycles the channel more than $6|S|$ times then there exists an infinite fair run.*

Proof. A run which cycles the channel more than $6|S|$ times (i.e. more times than there

are ranks) must remain in the same rank for a whole cycle since ranks are non-increasing during the run. The third property of Lemma 5.6 means that the channel cannot be cycled while remaining in the same "dead" rank — if the rank is $((Q,\Gamma),D)$ then there exists some $\alpha_i \notin \mathit{Reads}(Q,\Gamma)$ and no read transition between states in T is deleted from χ, so α_i cannot be read. Thus we can refine our observation to say that such a run must have the same "alive" rank for some whole cycle, say $((Q,\Gamma),A)$; say this cycle ends with $\beta_1 \dots \beta_l$ as the channel contents.

By the assignment process for ranks, we know that (Q,Γ) was a leaf node, and is therefore safe, thus $\mathit{Writes}(Q,\Gamma) \subseteq \mathit{Reads}(Q,\Gamma)$. Since the β_i were written by transitions in χ, we know $\beta_i \in \mathit{Reads}(Q,\Gamma)$ for all i. We can therefore extend the run from this point with a series of transitions from χ which enable us to read all the β_i and write only some $\gamma_1,\dots,\gamma_m$ where $\gamma_i \in \mathit{Reads}(Q,\Gamma)$. This extension process can be repeated infinitely to generate an infinite fair run. □

Having obtained this result, we observe that there are two ways for a channel machine $\mathcal{C}$ to have an infinite fair run. If $\mathcal{C}$ writes to the channel infinitely often, it must cycle the channel infinitely often and hence we need only consider runs of exactly $6|S|+1$ cycles by Proposition 5.5. Otherwise, $\mathcal{C}$ writes only finitely often, so it must reach a state from which it can loop while performing only read and *nop* operations. Since these transitions are always enabled due to the presence of insertion errors, we can ignore the channel for this part of the computation and compute a set F of states which, using only transitions in $\Delta' = \{(s,op,t) \in \Delta \mid op \in \{\sigma? \mid \sigma \in \Sigma\} \cup \{nop\}\}$, can reach such a loop by treating (S,Δ') as a nondeterministic automaton. Hence an infinite run with only finitely many writes can be considered to complete exactly a finite number of cycles (to end with an empty channel) and end in a state in F. As in the case above, we need only consider runs with at most $6|S|+1$ cycles by Proposition 5.5.

This type of cycle-bounded computation was explored in [BMOW07], where it was

(a) Computation Table (b) Computation Window

Figure 5.5: Deciding Cycle-Bounded Reachability

shown that the cycle-bounded reachability problem for insertion channel machines with renaming is decidable in space polynomial in the number of cycles. Their technique works by representing the computation as a table, with each row corresponding to one cycle and the operation of writing each letter vertically aligned with the subsequent read (as in Figure 5.5a). The empty spaces are filled by repeating the states which occur, leading to a picture as in Figure 5.5b. This figure also illustrates that we can use only the entries in a 3-space wide vertical window in order to verify the correctness of the transitions within — we can therefore decide whether a cycle-bounded computation which ends in a particular state exists by guessing an initial window, recording which states we guessed for the start of each cycle and updating the window by moving it one space to the right with each step (i.e. guessing a new final column). It is therefore clear that we only need space polynomial in the number of cycles in order to perform this nondeterministic construction of a run of the correct form. Note also that the alignment of writes with subsequent reads means that the fairness condition we impose is easy to verify.

We thus conclude this section with the following result:

Theorem 5.8. *The fair termination problem for insertion channel machines (without occurrence testing or renaming) is* PSPACE*-complete.*

5.4 Decidability of Fair Termination for ICMRs

In order to show the decidability of fair termination for insertion channel machines with renaming, we first simplify our model. We show that the renaming capability can be simulated by adding a single *reliable* symbol which is not subject to insertion errors. We then demonstrate that it is possible to "compress" such a machine $\mathcal{C}$, defining a machine $\mathcal{C}^n$ such that one cycle of $\mathcal{C}^n$ can simulate n cycles of $\mathcal{C}$. Finally, we consider the sets S_n of configurations from which $\mathcal{C}^n$ can complete one cycle and use Higman's Lemma to assert that the sequence $(S_n)_{n\geq 1}$ eventually stabilises. This implies that their intersection, the set of configurations from which an infinite fair run of $\mathcal{C}$ can originate, is computable.

5.4.1 Removing Renaming

For an insertion channel machine with renaming $\mathcal{C} = (S, s_0, \Sigma, \Delta)$, we wish to define a simulating machine $\mathcal{D}$, without renaming, such that $\mathcal{C}$ has an infinite fair run if and only if $\mathcal{D}$ has an infinite fair run.

Without loss of generality, we assume that there is a symbol $\triangleright \in \Sigma$ and that, intuitively, this symbol delineates cycles of $\mathcal{C}$. Formally, we assume that there is always a unique copy of $\triangleright$ on the channel (so it is written back to the channel immediately after being read and the renaming operations used do not affect $\triangleright$). Assume also that the only transitions available from s_0 have operand $\triangleright!$ to initialise this condition. Note that with these assumptions, any infinite fair run must read $\triangleright$ infinitely often (since we assume that it will be written immediately after every read and then fairness requires that it will later be read) and thus must contain infinitely many cycles.

We assume that $\mathcal{D}$ uses only operations from Op but can treat this $\triangleright$ symbol as *reliable* — it does not suffer from insertion errors. This condition can easily be simulated by using only the single occurrence test $zero(\triangleright)$ after every $\triangleright?$.

The idea is for $\mathcal{D}$ to store in its finite state two renaming relations along with a state of $\mathcal{C}$ — the first of these stores the composition of the renaming operations performed since reading $\triangleright$ and the second stores a guess of the composition of those which will be performed before $\triangleright$ is read again. Intuitively, when $\mathcal{D}$ reads a letter from the channel, it internally applies the first renaming which is an accumulation of those $\mathcal{C}$ would have applied since the beginning of the current cycle. When $\mathcal{D}$ writes a letter to the channel, it first applies the second renaming and writes the renamed letter to account for all of the renamings which $\mathcal{C}$ would have applied up to the end of the current cycle.

The states of $\mathcal{D}$ are therefore $(2^{\Sigma\times\Sigma_\epsilon} \times S \times 2^{\Sigma\times\Sigma_\epsilon})$, with initial state $(\emptyset, s_0, \emptyset)$. The channel alphabet Σ is unchanged from that of $\mathcal{C}$ and the transition relation Δ' is defined below by considering the types of transitions in Δ.

- If $(s_1, \sigma_1?, s_2) \in \Delta$, then $((L, s_1, R), \sigma?, (L, s_2, R)) \in \Delta'$ for all L, σ, R such that $\sigma L \sigma_1$. This type of transition simulates the machine $\mathcal{C}$ reading σ_1, having previously renamed σ to it. Note that if there is no σ_2 such that $\sigma_2 L \sigma_1$, all σ_1 letters have been renamed by $\mathcal{C}$ so this type of transition cannot simulate $\mathcal{C}$ reading σ_1 as an insertion error. Hence for each $(s_1, \sigma_1?, s_2) \in \Delta$ we also have $((L, s_1, R), nop, (L, s_2, R)) \in \Delta'$ for all L, R to account for $\mathcal{C}$ reading σ_1 as an insertion error.

- If $(t_1, \sigma_2!, t_2) \in \Delta$, then $((L, t_1, R), \sigma!, (L, t_2, R)) \in \Delta'$ for all L, σ, R such that $\sigma_2 R \sigma$. This type of transition simulates $\mathcal{C}$ writing σ_2 and renaming it to σ later in the cycle.

- If $(r_1, M, r_2) \in \Delta$, then $((L, r_1, R \circ M), nop, (M \circ L, r_2, R)) \in \Delta'$ for all L, R. This type of transition moves M from being a renaming we have guessed will apply in the future to one we have performed already.

- If $(q_1, \triangleright?, q_2) \in \Delta$, then $((L, q_1, R), \triangleright?, (L, q_2, R)) \in \Delta'$ for all L, R. We simulate the $\triangleright?$ transition directly and use the following $\triangleright!$ transition to prepare for the next cycle.

- If $(p_1, \triangleright!, p_2) \in \Delta$, then $((L, p_1, id), \triangleright!, (id, p_2, R)) \in \Delta'$ for all L, R. The $\triangleright!$ transition at the end of the cycle is where we verify that our guess of the upcoming renaming operations was correct and is therefore only enabled when this component of the state is the identity relation. It resets our guess of the transitions to be performed in the next cycle to R.

- Finally, we need some initialisation transitions not included in Δ; $((\emptyset, s_0, \emptyset), nop, (id, s_0, R)) \in \Delta'$ for all R. This transition sets our guess of the renamings which will be applied over the course of the first cycle to R.

Lemma 5.9. *There exists an infinite fair run of $\mathcal{C}$ on empty input if and only if there exists an infinite fair run of $\mathcal{D}$ on empty input.*

Proof. Suppose there exists an infinite fair run $(s_0, \emptyset) \xrightarrow{\triangleright!} (s_1, \triangleright) \xrightarrow{o_1} (s_2, u_2) \xrightarrow{o_2} \cdots$ of $\mathcal{C}$. Since we assume without loss of generality that there always exists a unique copy of $\triangleright$ on the channel during this run, we know that it must contain infinitely many cycles (as the fairness condition ensures that $\triangleright$ is read and then written infinitely often).

We match the initial transition $(s_0, \emptyset) \xrightarrow{\triangleright!} (s_1, \triangleright)$ in the run of $\mathcal{C}$ with the sequence of transitions $((\emptyset, s_0, \emptyset), \emptyset) \xrightarrow{nop} ((id, s_0, id), \emptyset) \xrightarrow{\triangleright!} ((id, s_1, R), \triangleright)$ of $\mathcal{D}$ for some guess R of the composition of the renamings in the second cycle.

Then we show that for each cycle

$$\xrightarrow{\triangleright!} (s_i, u_i) \xrightarrow{o_i} \cdots (s_{i+k}, u_{i+k}) \xrightarrow{\triangleright!} (s_{i+k+1}, u_{i+k+1})$$

in this run of $\mathcal{C}$, there exists a cycle

$$\xrightarrow{\triangleright!} ((id, s_i, R_1), u_i) \xrightarrow{o_i} \cdots ((L, s_{i+k}, id), u_{i+k}) \xrightarrow{\triangleright!} ((id, s_{i+k+1}, R_2), u_{i+k+1})$$

of $\mathcal{D}$ (for some R_1, R_2 and L) where each transition $(s_{i+j}, u_{i+j}) \xrightarrow{o_{i+j}} (s_{i+j+1}, u_{i+j+1})$ for $0 \leq j < k$ of $\mathcal{C}$ is matched by a single transition of $\mathcal{D}$.

Suppose that the sequence of renaming operations in this cycle of $\mathcal{C}$ is $M_1, M_2, \ldots, M_n$ and let $R = M_n \circ M_{n-1} \cdots \circ M_2 \circ M_1$. By our definition of the transitions of $\mathcal{D}$ above, it is clear that the required transition $\xrightarrow{\triangleright!} ((id, s_i, R), u_i)$ at the start of the cycle is possible.

Next we show how $\mathcal{D}$ matches the transition $(s_{i+j}, u_{i+j}) \xrightarrow{o_{i+j}} (s_{i+j+1}, u_{i+j+1})$ of $\mathcal{C}$, supposing that the renaming operations $M_1, \ldots, M_l$ have been performed by the previous transitions. We assert that $\mathcal{D}$ has reached the configuration $((L_j, s_j, R_j), v_{i+j})$ where $L_j = M_1 \circ M_2 \cdots \circ M_l$, $R_j = M_n \circ M_{n-1} \cdots \circ M_{l+1}$ and if $u_{i+j} = w \cdot \triangleright \cdot x$, $v_{i+j} = y \cdot \triangleright \cdot z$ where yL_jw and xR_jz then show that the transition of $\mathcal{D}$ maintains this invariant. There are several cases:

- If $o_{i+j} = \sigma_1?$ and $u_{i+j} = \sigma_1 \cdot u_{i+j+1}$, $\mathcal{D}$ performs the transition

$$((L_j, s_j, R_j), \sigma \cdot v_{i+j+1}) \xrightarrow{\sigma?} ((L_j, s_{j+1}, R_j), y_{i+j+1})$$

  where $\sigma L_j \sigma_1$. This transition exists in $\mathcal{D}$ by our definition of Δ' and the fact that yL_jw; it is clear that the new values of w and y respectively have each had a single letter removed, so this condition is maintained and the other conditions of the invariant are unaffected.

- If $o_{i+j} = \sigma_1?$ and $u_{i+j} = u_{i+j+1}$, $\mathcal{D}$ performs the transition

$$((L_j, s_j, R_j), v_{i+j}) \xrightarrow{nop} ((L_j, s_{j+1}, R_j), v_{i+j}) .$$

  This transition exists in $\mathcal{D}$ by our definition of Δ'. It is clear that the conditions of the invariant are unaffected.

- If $o_{i+j} = \sigma_2!$, $\mathcal{D}$ performs the transition

$$((L_j, s_j, R_j), v_{i+j}) \xrightarrow{\sigma!} ((L_j, s_{j+1}, R_j), v_{i+j} \cdot \sigma)$$

  where $\sigma_2 R_j \sigma$. This transition exists in $\mathcal{D}$ by our definition of Δ' and the fact that xR_jz; it is clear that the new values of z and x respectively have each had a single

letter added, so this condition is maintained and the other conditions of the invariant are unaffected.

- If $o_{i+j} = M_{l+1}$, $\mathcal{D}$ performs the transition

$$((L_j, s_j, R_j), v_{i+j}) \xrightarrow{nop} ((L_{j+1}, s_{j+1}, R_{j+1}), v_{i+j})$$

where $L_{j+1} = L_j \circ M_{l+1}$ and $R_j = R_{j+1} \circ M_{l+1}$. This transition exists in $\mathcal{D}$ by our definition of Δ'. Then $u_{i+j+1} = M_{l+1}(w) \cdot \triangleright \cdot M_{l+1}(x)$ since we assume that all renaming operations leave $\triangleright$ unaffected. Since yL_jw, we have that $y(M_{l+1} \circ L_j)(M_{l+1}(w))$, that is $yL_{j+1}(M_{l+1}(w))$; since xR_jz, we have that $x(R_{j+1} \circ M_{l+1})z$, that is $xR_{j+1}(M_{l+1}(z))$. Hence all the conditions of our invariant are maintained.

This completes the proof that we can match the transitions within a cycle of $\mathcal{C}$, hence it completes the only-if direction of the proof of the lemma; any infinite fair run of $\mathcal{C}$ corresponds to an infinite fair run of $\mathcal{D}$. It is simple enough to reverse this argument to verify the other direction. □

Our intuition is thus that a single reliable symbol $\triangleright$ is equivalent in expressive power to a full set of renaming operations, even without using any occurrence test operations. More formally, we could use occurrence tests only on $\triangleright$ to ensure that it does not suffer from insertion errors as our intuitive model requires. We hence assume in the following that our insertion channel machines treat this $\triangleright$ symbol as reliable but use only operations from Op, i.e. no renaming or occurrence tests.

5.4.2 Cycle Compression

For an insertion channel machine $\mathcal{C} = (S, s_0, \Sigma, \Delta)$ which uses operations from Op only but treats the symbol $\triangleright \in \Sigma$ as reliable, we define the product $\mathcal{C}^n$ which intuitively compresses n cycles of computation of the machine $\mathcal{C}$ into one. There exists a run of $\mathcal{C}^n$ which begins

in configuration (s, w), ends in configuration (t, v) and cycles the channel once if and only if there exists a run of $\mathcal{C}$ which begins in configuration (s, w), ends in configuration (t, v) and cycles the channel n times.

Recall our assumption that that there is always a unique copy of $\triangleright$ on the channel (so it is written back to the channel immediately after being read and the only transitions available from s_0 have operation $\triangleright!$ to initialise this condition). We can hence measure the number of cycles of the channel by the number of $\triangleright!$ operations performed.

We then define

$$\mathcal{C}^n \stackrel{\text{def}}{=} (S \times (S \cup \{\bot\})^{2(n-1)}, (s_0, \bot^{2(n-1)}), \Sigma, \Delta')$$

where the states store $2n-1$ copies of a state of $\mathcal{C}$. The first n are used to simulate n copies of $\mathcal{C}$ and the remainder are used for verification. The special states tagged by $\bot$ are used to initialise the simulation with a guess of the initial state of the $n-1$ copies of $\mathcal{C}$ other than the first. The initial state is therefore essentially unchanged, as is the alphabet. The transition relation Δ' is defined by the following:

- If $(r_1, \sigma?, r_2) \in \Delta$ for $\sigma \in \Sigma$, then

  $$((r_1, t_1, \ldots, t_{n-1}, g_1, \ldots, g_{n-1}), \sigma?, (r_2, t_1, \ldots, t_{n-1}, g_1, \ldots, g_{n-1})) \in \Delta'$$

  for all $t_1, \ldots, t_{n-1}$, $g_1, \ldots, g_{n-1}$. This type of transition simulates the machine reading σ in the first cycle. We also have

  $$\left(\begin{array}{ll} (s, t_1, \ldots, t_{i-2}, r_1, t_i, \ldots, t_{n-1}, g_1, \ldots, g_{n-1}), & nop, \\ (s, t_1, \ldots, t_{i-2}, r_2, t_i, \ldots, t_{n-1}, g_1, \ldots, g_{n-1}) & \end{array} \right) \in \Delta'$$

  for all s, $t_1, \ldots, t_{i-2}$, $t_i, \ldots, t_{n-1}$, $g_1, \ldots, g_{n-1}$ and $2 \leq i \leq n$. This type of transition simulates the machine reading σ in the ith cycle as an insertion error.

- If $(w_1, \sigma!, w_2) \in \Delta$ for $\sigma \in \Sigma \setminus \{\triangleright\}$, then

  $$((s, t_1, \ldots, t_{n-2}, w_1, g_1, \ldots, g_{n-1}), \sigma!, (s, t_1, \ldots, t_{n-2}, w_2, g_1, \ldots, g_{n-1})) \in \Delta'$$

for all $s,\ t_1, \ldots, t_{n-2},\ g_1, \ldots, g_{n-1}$. This type of transition simulates the machine writing σ in the nth cycle. Note that we exclude writes of $\triangleright$ in the nth cycle here as we deal with them separately.

- If $(r_1, \sigma?, r_2), (w_1, \sigma!, w_2) \in \Delta$ for $\sigma \in \Sigma$, then

$$\left(\begin{array}{ll} (t_1, \ldots, t_{i-1}, w_1, r_1, t_{i+2}, \ldots, t_n, g_1, \ldots, g_{n-1}), & nop, \\ (t_1, \ldots, t_{i-1}, w_2, r_2, t_{i+2}, \ldots, t_n, g_1, \ldots, g_{n-1}) & \end{array} \right) \in \Delta'$$

for all $t_1, \ldots, t_{i-1},\ t_{i+2}, \ldots, t_n,\ g_1, \ldots, g_{n-1}$ and $1 \leq i < n$. This type of transition simulates the machine writing σ in the ith cycle and it being read in the $i+1$st cycle. Note that our fairness requirement means that every symbol written in the ith cycle must be read in the $i+1$st.

- If $(p_1, \triangleright!, p_2) \in \Delta$, then

$$((g_1, \ldots, g_{n-1}, p_1, g_1, \ldots, g_{n-1}), \triangleright!, (p_2, t_1, \ldots, t_{n-1}, t_1, \ldots, t_{n-1})) \in \Delta'$$

for all $g_1, \ldots, g_{n-1}$ and $t_1, \ldots, t_{n-1}$. The $\triangleright!$ transition at the end of the nth cycle is where we verify that our guesses of the state occupied at the end of the each previous cycle are correct. Having simulated n cycles of $\mathcal{C}$ that ended in state p_1, we simulate n more cycles by starting our first copy of $\mathcal{C}$ in state p_2; we moreover guess that the other $n-1$ copies of $\mathcal{C}$ will begin their cycles in locations $t_1, \ldots, t_{n-1}$ respectively.

- If $(r_1, nop, r_2) \in \Delta$, then

$$\left(\begin{array}{ll} (t_1, \ldots, t_{i-1}, r_1, t_i, \ldots, t_n, g_1, \ldots, g_{n-1}), & nop, \\ (t_1, \ldots, t_{i-1}, r_2, t_i, \ldots, t_n, g_1, \ldots, g_{n-1}) & \end{array} \right) \in \Delta'$$

for all $t_1, \ldots, t_{i-2},\ t_i, \ldots, t_{n-1},\ g_1, \ldots, g_{n-1}$ and $1 \leq i \leq n$. This type of transition allows the machine to perform a *nop* operation in any of the n cycles we simulate.

- Finally, we need some initialisation transitions not included in Δ;

$$\big((s_0, \bot^{2(n-1)}), nop, (s_0, t_1, \ldots, t_{n-1}, t_1, \ldots, t_{n-1})\big) \in \Delta'$$

for all $t_1, \ldots, t_{n-1}$. This transition sets our initial guess of the location the machine $\mathcal{C}$ will be in at the end of each of the cycles 1 to $n-1$ (and hence the beginning of cycles 2 to n).

5.4.3 Higman's Lemma

In this section, we provide an algorithm for deciding the fair termination problem for insertion channel machines with renaming by computing for each k in turn the language L_k of initial configurations from which such a machine $\mathcal{C}$ can perform at least k cycles. We introduce the theory of *well-quasi-orderings* and use this to demonstrate that as k increases, this sequence of languages eventually stabilises (to the set of initial configurations from which $\mathcal{C}$ has an infinite fair run) and hence our computation terminates. This process follows the techniques of Finkel and Schnoebelen [FS01] closely, but does not map exactly to their general framework of *well structured transition systems*.

Definition 5.10. A *quasi-ordering* (qo) is a reflexive and transitive relation $\leq$.

We let $x < y$ denote $x \leq y$ and $y \not\leq x$.

Definition 5.11. A *well-quasi-ordering* (wqo) is any qo $\leq$ (over some set X) such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X, there exist indices $i < j$ with $x_i \leq x_j$.

Note that any wqo is *well-founded* — it admits no infinite strictly decreasing sequence $x_0 > x_1 > x_2 > \ldots$

Given $\leq$ a qo, a downward-closed set is any set $I \subseteq X$ such that $y \leq x$ and $x \in I$ entail $y \in I$. To any $x \in X$ we associate $\downarrow x \stackrel{\text{def}}{=} \{y \mid y \leq x\}$, which is the smallest downward-closed set containing x.

Lemma 5.12. *If $\leq$ is a wqo, any infinite decreasing sequence $I_0 \supseteq I_1 \supseteq I_2 \supseteq \cdots$ of downward-closed sets eventually stabilises, i.e. there is a $k \in \mathbb{N}$ such that $I_k = I_{k+1} = I_{k+2} = \cdots$.*

Proof. This result follows directly from the definition of a wqo. Suppose for a contradiction that we have an infinite subsequence where the inclusion is strict, i.e. $I_{n_0} \supsetneq I_{n_1} \supsetneq I_{n_2} \supsetneq \cdots$. Then there exists an infinite sequence of elements x_i in $I_{n_i} \setminus I_{n_{i+1}}$. From the definition of a wqo, there exist $i < j$ such that $x_i \leq x_j$. Since I_{n_j} is downward closed, $x_i \in I_{n_j}$, contradicting the fact that the inclusion is strict. □

We now introduce the *subword* ordering $\sqsubseteq$ over the set of finite words, which is defined as follows:

For words $\rho = \rho_1\rho_2\ldots\rho_m$ and $\mu = \mu_1\mu_2\ldots\mu_n$, $\rho \sqsubseteq \mu$ if there exists an injective monotonic function f such that $\rho_i = \mu_{f(i)}$ for all $i \leq m$. Thus this ordering requires that the letters of ρ are contained in the letters of μ in the correct sequence, though not necessarily consecutively. For example, HIM $\sqsubseteq$ HIGMAN.

Lemma 5.13. *(Higman [Hig52]) The subword ordering $\sqsubseteq$ is a wqo.*

Using this subword ordering, we define an ordering on the configurations of a channel machine $\mathcal{C}$ as follows:

Definition 5.14. $(s, x) \leq_{\mathcal{C}} (t, y)$ if and only if $s = t$ and $x \sqsubseteq y$.

Lemma 5.15. *For every channel machine $\mathcal{C}$, the ordering $\leq_{\mathcal{C}}$ is a wqo.*

Proof. It is clear that $\leq_{\mathcal{C}}$ is reflexive and transitive and thus a quasi-order.

Now consider an infinite sequence $(s_0, x_0), (s_1, x_1), (s_2, x_2), \ldots$. There exist only finitely many states of $\mathcal{C}$, so there exists an infinite subsequence $(s_{i_0}, x_{i_0}), (s_{i_1}, x_{i_1}), (s_{i_2}, x_{i_2}), \ldots$ where $s_{i_0} = s_{i_1} = s_{i_2} = \cdots$. Since $\sqsubseteq$ is a wqo there exist some indices j, k with $j < k$ and $x_{i_j} \sqsubseteq x_{i_k}$. Hence $(s_{i_j}, x_{i_j}) \leq_{\mathcal{C}} (s_{i_k}, x_{i_k})$ and thus $\leq_{\mathcal{C}}$ is a wqo. □

Using these results, we are now able to prove the main theorem of this section.

Theorem 5.16. *The fair termination problem is decidable for insertion channel machines with renaming.*

Proof. For $\mathcal{C}$ an insertion channel machine with renaming, we first apply the construction of Section 5.4.1 to produce an insertion channel machine $\mathcal{D}$ with reliable end marker $\triangleright$ such that $\mathcal{D}$ has an infinite fair run with empty input if and only if $\mathcal{C}$ does. We then use the product construction of Section 5.4.2 to define $\mathcal{D}^k$ for each k.

We define L_k to be the set of configurations from which $\mathcal{D}$ has a run which performs k $\triangleright!$ operations. By our assumptions that $\triangleright$ only appears once on the channel and can only be written after first being read, this means that $\mathcal{D}$ must cycle the channel k times. It is clear that this set is downward closed in the ordering $\leq_{\mathcal{D}}$ since if $\mathcal{D}$ can complete k cycles starting from state s and channel contents x and $y \sqsubseteq x$, it can do so from state s and channel contents y (by reading the missing letters of x as insertion errors). It is similarly clear that $L_{k+1} \subseteq L_k$ since any run of the channel machine $\mathcal{D}$ which lasts for $k+1$ cycles also lasts k cycles.

Hence by Lemma 5.12, the sequence $L_1 \supseteq L_2 \supseteq L_3 \supseteq \cdots$ eventually stabilises, say to L_N. If $(s_0, \emptyset) \notin L_N$, then $\mathcal{D}$ has no fair run of at least N cycles with empty input, so $\mathcal{D}$ has no infinite fair run with empty input.

However if $(s_0, \emptyset) \in L_N$, we show that we can construct an infinite fair run of $\mathcal{D}$. Since $L_1 \supseteq L_N$, we know that there exists a run of $\mathcal{D}$ that completes one cycle, ending in some configuration, say (s_1, x_1). Moreover, since $L_N = L_{N+1}$, we can choose (s_1, x_1) such that $(s_1, x_1) \in L_N$ and can repeat this process, extending the run for an additional cycle to some $(s_2, x_2) \in L_N$. We can repeat this process of extending the run by one cycle infinitely, thus we obtain a run with infinitely many cycles, which must be fair as observed in Section 5.2.3.

We now explain how to compute L_k for any k. Instead of trying to consider k cycles of the machine $\mathcal{D}$, we can instead consider one cycle of the machine $\mathcal{D}^k$; if $(s, x) \in L_k$, then $\mathcal{D}$

can complete k cycles from initial state s, so $\mathcal{D}^k$ can complete one cycle from initial state $(s, \bot^{2(k-1)})$ by the definition of $\mathcal{D}^k$. Moreover, since the letters written to the channel have no impact on this property, we can compute this set by treating $\mathcal{D}^k$ as a nondeterministic finite automaton — we replace all transitions with operand $\sigma!$ by ones labelled *nop*, a new accepting state as the target of all $\triangleright!$ transitions and a new initial state, which has one transition labelled by each state s of $\mathcal{D}$ which moves the automaton to $(s, \bot^{2(k-1)})$.

By the reasoning above, we need only compute finitely many L_k, hence a procedure to compute each in turn will terminate. □

We now consider the complexity of this procedure and recall the *Grzegorczyk hierarchy* [Grz53] of fast-growing functions, which is defined as follows:

$$F_0(n) = n + 1$$

$$F_{k+1}(n) = F_k^{n+1}(n) = \overbrace{F_k(F_k(\ldots F_k}^{n+1 \text{ times}}(n)\ldots))$$

Further, recall the family of $\exp_k$ functions from Section 3.6.2 which are defined inductively by $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$, so

$$\exp_k(n) = 2^{\cdot^{\cdot^{\cdot^{2^n}}}\Big\}k}$$

We observe that $F_1(n) = 2n + 1$, $F_2(n) = (n + 1)2^{n+1} - 1$, $F_3(n) = \Theta(\exp_n(1))$. This hierarchy has been extended to ordinals by Löb and Wainer [LW70] and this extension places the Ackermann function at F_ω. We believe that one could apply recent techniques of Schnoebelen [SS11], which were developed for analysis of the reachability problem for lossy channel machines [CS08], in order to determine the complexity of our decision procedure for the fair termination problem. We conjecture that these techniques would yield a bound of F_{ω^ω}, but note that such a bound would not even be multiply recursive. In the next section we investigate a lower bound on the complexity of fair termination, but cannot exclude the possibility of a primitive recursive decision procedure.

5.5 Hardness of Fair Termination for ICMOTs

In this section we describe how to use an insertion channel machine with occurrence testing to represent a counter and detect when it suffers an insertion error. We represent counters using sequences of 0 and 1 binary bits and design our channel machine so that, in the absence of errors, all operations on these bits are *length preserving*: whenever a counter bit is read from the channel, one is immediately written to replace it. We will therefore detect insertion errors by monitoring the length of these counters — all insertion errors would increase the length of the bit sequence representing the counter.

Since a binary counter of length l can count up to $2^l - 1$, we can follow the yardstick construction of Stockmeyer and Meyer [SM73] which employs a series of progressively longer counters, using each to ensure the length of the next remains correct. Our construction is parametric in the size of the alphabet used; a larger alphabet allows us to maintain more counters. In the following, we show that with $6k + 3$ letters and $O(nk)$ states, we can construct an insertion channel machine with occurrence testing which maintains a counter of maximum value $O(F_3^k(n))$.

Throughout this section we use the channel letter $\triangleright$ exclusively as an *end marker*. We require that there always be exactly one occurrence of $\triangleright$ on the channel at any time, maintaining this invariant by always performing the occurrence test $zero(\triangleright)$ and then writing $\triangleright$ immediately after each read of $\triangleright$. This invariant means that we count the number of cycles in a run of an insertion channel machine with occurrence testing by the number of times it has read (or written) $\triangleright$.

5.5.1 A Small Perfect Counter

It is clear that for any fixed N we can use the states of the machine to store the state of a counter of maximum value N. We now show how to use a counter of maximum value N

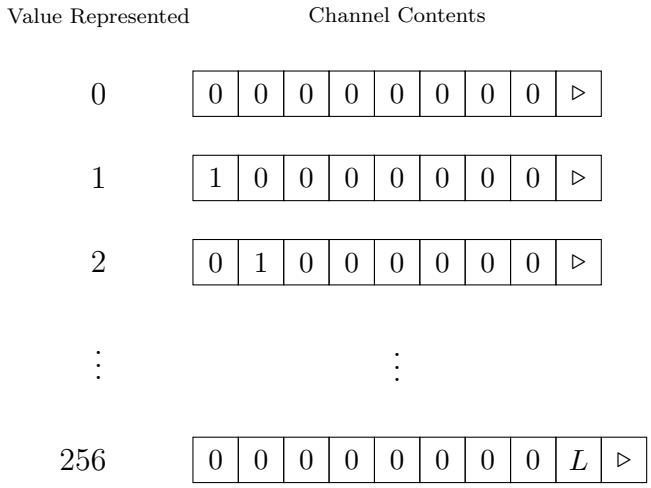


Figure 5.6: Values are least significant bit first; L indicates a maximum value

stored in the channel machine's states to ensure that a counter of *length* N, and therefore maximum value 2^N, stored on the channel remains perfect.

We represent the counter in binary as a length-N sequence of 0 or 1 symbols, with the least significant bit first. We represent the value 2^N by a sequence of N zeros followed by a special symbol L (standing for *lock*), which indicates that the counter cannot be incremented again before it is zero-tested; the zero-test operation accepts either the counter value 0 or 2^N and halts otherwise. Examples of how these values are stored on the channel are displayed in Figure 5.6.

To ensure the integrity of the channel counter, we zero the state counter before performing any operations on the channel counter, increment it whenever a counter symbol (0 or 1) is read and test that when the state counter has value N, the symbol $\triangleright$ (which marks the end of the channel counter) has been reached. This detects insertion errors because they would always cause the channel counter to have a length greater than N.

To perform a zero test, we simply follow this process reading only 0 symbols, as illustrated in Figure 5.7. Note that the $L?$ operation does not block execution in the case that the counter represents 0 (and therefore has no L symbol) as the machine may read an

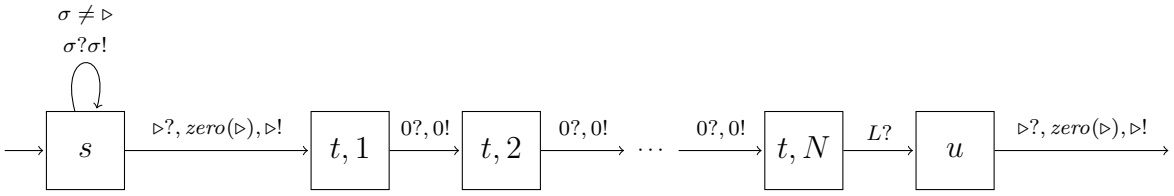


Figure 5.7: Zero-Test

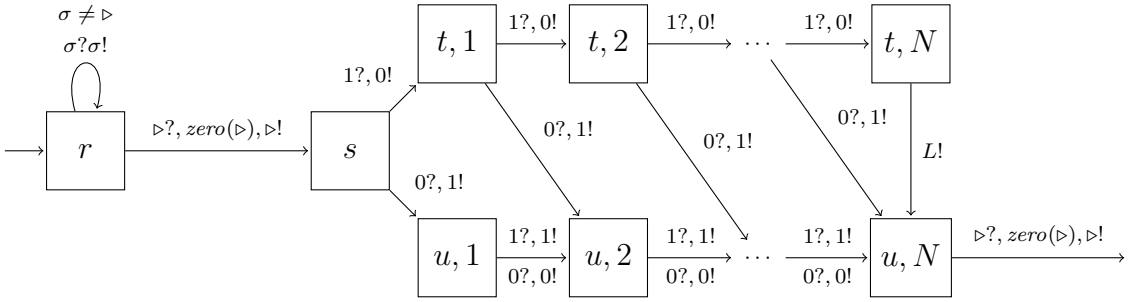


Figure 5.8: Increment

insertion error here.

To perform an increment, we flip bits until a 0 is encountered then just copy the remaining bits of the channel — this process is illustrated in Figure 5.8. The corresponding process for decrements is similar.

5.5.2 Inductive Sequences of Counters

In this section, we show how to use the channel of an insertion channel machine with occurrence testing to store a sequence of binary counters of increasing length, using each counter to verify the length of the next counter remains correct and thus detecting insertion errors.

We name these counters C_i for $0 \leq i < n$, using n hash symbols to divide them. We assume that counter C_0 is a perfect counter of length N, using the states of the channel machine to detect insertion errors as described in Section 5.5.1. Since there are many

such counters we wish to remember which operation each of them is currently performing by using different *state symbols* rather than the states of the machine. We therefore use state symbols # to denote no operation, $\#_I^1$ and $\#_I^2$ for an increment, $\#_D^1$ and $\#_D^2$ for a decrement and $\#_Z$ for a zero test. We always wish to manipulate the smallest counter which has a current operation, so these state symbols will implement a call stack. We also use a $\downarrow$ symbol to record the position we have reached in each counter.

The counter C_i is represented by a binary sequence of $\exp_i(N)$ 0 or 1 symbols with the least significant bit first, preceded by a state symbol. Just as in Section 5.5.1 above, we represent the maximum value $\exp_{i+1}(N)$ of this counter by a sequence of zeros followed by the special symbol L. Again we allow the zero-test operation to accept either the value 0 or $\exp_{i+1}(N)$ and require it to halt otherwise.

In order to use this sequence of counters, we crucially exploit the nondeterminism available to channel machines — whenever we wish to interact with a counter it will intuitively be the counter atop the call stack i.e. it will be the last counter which is preceded by the unannotated # symbol (which denotes no current operation), so we simply guess which one that is. It is clear that if we guess a counter other than the last, we can easily verify this guess as we continue to read the channel, halting in case we read # (i.e. our guess was wrong). If we guess that no counter has a current operation, the $\triangleright$ symbol at the end of the channel allows us to verify that we have considered each in turn.

We use this scheme to reason about interacting with the counter C_i in the course of acting on the counter C_{i+1} (where we are not acting on any lower counter). If we record our place within C_{i+1} with the symbol $\downarrow$, we can leave our work in this counter, nondeterministically guess which counter is C_i — the last without an operation, as explained above — operate on it and return to the correct position within C_{i+1} when we are finished.

To ensure the integrity of the counter C_{i+1}, we zero the counter C_i before performing any operations on counter C_{i+1}, increment C_i whenever a counter symbol (0 or 1) of C_{i+1}

is read and check that when the end of C_{i+1} is reached, C_i has maximum value using a zero-test. This detects insertion errors in C_{i+1} because they would always increase the length of C_{i+1} and prevent C_i from performing one of the required increments.

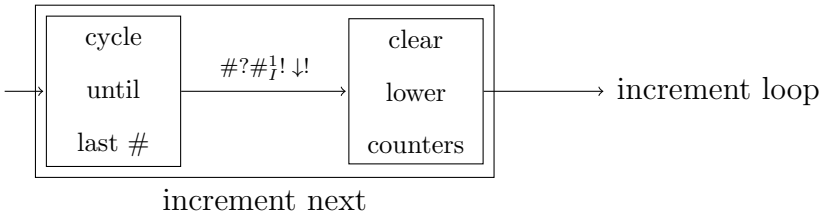


Figure 5.9: The *increment next* sub-operation

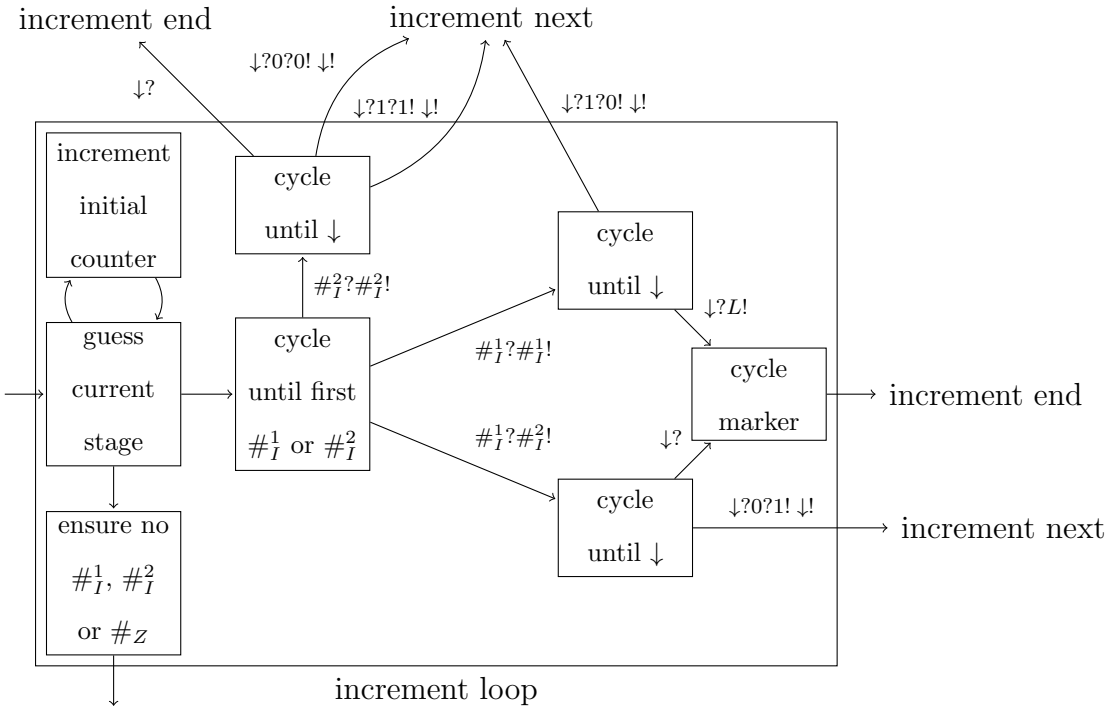


Figure 5.10: The *increment loop* sub-operation

We consider that the increment operation shown in three parts in Figures 5.9, 5.10 and 5.11 acts solely on the largest counter C_N, using the other counters only for the integrity tests mentioned above. During the execution of this operation, we maintain the

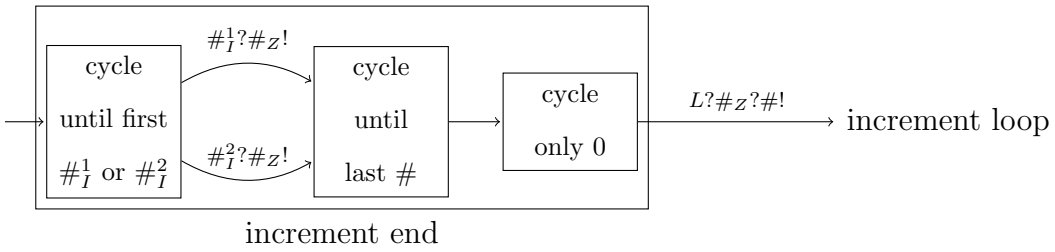


Figure 5.11: The *increment end* sub-operation

invariant that for each i, if C_{i+1} contains the $\downarrow$ symbol, the number of 0 or 1 symbols of C_{i+1} which precede the $\downarrow$ is equal to the binary value represented by the counter C_i. This operation begins with the sub-operation *increment next*, before proceeding to the *increment loop* and finally *increment end* sub-operations. We abbreviate the many places in which the machine cycles the channel around, writing back each symbol as it is read. In some places, we cycle any single state symbol or $\triangleright$; these are abbreviated "cycle marker", in the others we simply cycle the channel until we nondeterministically guess that we have reached a particular symbol on the channel.

Intuitively, the increment operation begins by placing the $\#_I^1$ and $\downarrow$ symbols at the head of the last (before the $\triangleright$ symbol) counter, changes all lower counters to be zero (by rewriting any 1 symbols), and then begins a loop until there are no $\#_I$ or $\#_Z$ symbols left on the channel.

In the loop, we cycle around to the first counter with a $\#_I$ symbol. If that is the initial counter, we increment it in a single pass using the states of the machine to check it has the right length (as described in Section 5.5.1) and return to the beginning of the loop.

Otherwise, it is a higher counter, say C_{i+1}; in this case, we find the $\downarrow$ symbol on that counter and advance it one place, updating it and the 0 or 1 symbols as required. We then begin the initialisation which causes the counter C_i to be incremented by placing $\#_I^1$ and $\downarrow$ at its head before returning to the beginning of the loop.

If we instead reach the end of the counter C_{i+1}, we replace the $\#_I$ symbol at the start of this counter with $\#_Z$ then verify that the preceding counter C_i contains only 0 symbols (i.e. C_{i+1} has the correct length), replace the $\#_Z$ at the head of C_{i+1} with # and return to the beginning of the loop.

When the loop finally exits, there are no more $\#_I$ or $\#_Z$ symbols on the channel so we are sure that we have correctly incremented the last counter; insertion errors of 0 or 1 symbols are detected by the length checking and insertions of lone state symbols would be detected because we would perform more increments than necessary, a case prevented by the L symbols. The only error that we cannot detect is the insertion of an entire extra counter (of the correct size), with suitable adjustments to the size of the other counters — we cannot detect this because we have not specified any way to determine the correct number of counters (we again need to count, this time # symbols).

The zero-test and decrement operations are implemented similarly.

5.5.3 Large Perfect Counters

The procedure described in the previous section for inductively constructing large counters stored on the channel of an insertion channel machine is able to ensure that each counter in the construction is exponentially longer than the last, but has no way to determine the number of such counters on the channel. In this section, we describe how to remedy this deficit to construct a perfect counter with maximum value $O(F_3^k(n))$ with an insertion channel machine with occurrence testing using $6k + 3$ letters and $O(nk)$ states.

Initially, we use the states of the channel machine to keep track of the number of counters we store on the channel by counting the number of # symbols, either plain or annotated. Assuming that we choose to use k such counters, the first of which has length n (also ensured by the states of the machine), the last counter will be able to count as high as $\exp_k(n)$.

We then use this final counter to verify the length of a sequence of $\flat$ state symbols stored at the end of the channel (i.e. $\flat$, $\flat_I^1$, $\flat_I^2$, $\flat_D^1$, $\flat_D^2$ and $\flat_Z$). With appropriate extra states and exchanges of $\flat$ for # symbols in the above, we can use the same inductive procedure to ensure that a second sequence of counters of length $\exp_k(n)$ remains insertion error free. Again assuming the first of these has length n, the last will count as high as $\exp_{\exp_k(n)}(n)$.

In principle, we can continue this expansion indefinitely at the cost of 6 extra letters in the channel alphabet and a polynomial increase in the size of the set of states and transitions required.

Moreover, we can use the final counter to ensure that a sequence of symbols representing the cells of a Turing machine's tape are guarded against insertion errors. Hence we can directly simulate a Turing machine with state space of size N, alphabet of size A, input of length n and space bound $\exp_n(1) = O(F_3(n))$ with a channel machine of size polynomial in N and n and alphabet of size $A + 11$. We can also directly simulate a Turing machine with space bound $\exp_{\exp_n(1)}(1) = O(F_3(F_3(n)))$ with a channel machine of size polynomial in N and n and alphabet of size $A + 17$. Since we can repeat this process a number of times linear in A while the size of the channel machine remains polynomial in N and n, we conclude this section with the following theorem since $F_4(n) = F_3^{n+1}(n)$.

Theorem 5.17. *Given a Turing machine $\mathcal{M}$ and input I of length n, if $\mathcal{M}$ has space bound $F_4(n)$, we can construct (in space logarithmic in $|\mathcal{M}|$ and n) an insertion channel machine with occurrence testing $\mathcal{C}$ of size polynomial in $|\mathcal{M}|$ and n such that $\mathcal{C}$ has an infinite fair run if and only if $\mathcal{M}$ has a non-halting computation on I.*

The claim that this construction can be performed in space logarithmic in $|\mathcal{M}|$ and n is justified by the observation that the construction generates $n + 1$ sequences of counters plus a copy of $\mathcal{M}$; each sequence of counters is composed of a fixed number of states and transitions which can be constructed by (re)using a constant number of pointers to elements of $\mathcal{M}$ and I.

5.6 Safety MTL

Having established that the fair termination problem for insertion channel machines with renaming is decidable but SPACE($F_4(n)$)-hard, we use the rest of this chapter to show that these results can equally be applied to problems of temporal logic and alternating timed automata.

We first focus on the connection with temporal logic, specifically the Safety fragment of Metric Temporal Logic. After recalling the definition of this logic, we use this section to demonstrate that the fair termination problem for insertion channel machines with renaming reduces to the satisfiability problem for Safety MTL.

Temporal properties which are useful in verification can often be thought of as falling into one of two intuitive categories — *safety* properties, which require that "something bad" never happens and *liveness* properties, which require that "something good" eventually happens [Lam77]. These categories were formalised by the semantic definitions of Alpern and Schneider [AS85] as follows: a property P is a *safety property* if for every infinite timed word ρ which does not satisfy P, there exists a finite *bad prefix* ρ_i of ρ such that every extension μ of σ_i to an infinite timed word $\rho_i\mu$ does not satisfy P; P is a *liveness property* if for every finite timed word ρ_i there exists an infinite timed word μ such that the infinite timed word $\rho_i\mu$ satisfies P. Alpern and Schneider went on to show that using these definitions, every property is the intersection of a liveness property and a safety property [AS85]. Note, however, that these semantic definitions may not always follow our intuition as every unsatisfiable property is a safety property (with any value for the prefix length i). An example of such an unsatisfiable property which does not meet our intuition of safety would be "always $\neg p$ and eventually p", since this seems to have a liveness component.

Safety MTL is a syntactic rather than semantic fragment of MTL and as such is conservative with respect to these semantic definitions — all properties expressible by Safety

MTL are safety properties, however not every unsatisfiable MTL formula is in Safety MTL.

5.6.1 Defining Safety MTL

Recall from Section 2.5 that the logic MTL extends the classical linear time temporal logic LTL with time constraints on the *next* ($\bigcirc_I \varphi$), *until* ($\varphi_1 \mathbf{U}_I \varphi_2$) and *dual until* ($\varphi_1 \widetilde{\mathbf{U}}_I \varphi_2$) modalities. In each case I is an interval with endpoints in $\mathbb{N} \cup \{\infty\}$, though we often omit I when we refer to the interval $[0, \infty)$ and use arithmetic constraints to refer to these intervals (such as ≤ 1 to refer to $[0, 1]$). We use $\top$ and $\bot$ as the usual *true* and *false* symbols and define syntactic shorthands for the modalities *eventually*: $\Diamond_I \varphi \stackrel{\text{def}}{=} \top \mathbf{U}_I \varphi$ and *always*: $\Box_I \varphi \stackrel{\text{def}}{=} \bot \widetilde{\mathbf{U}}_I \varphi$.

Recall that a *timed word* over an alphabet Σ is a pair (σ, τ) where $\sigma = \sigma_1 \sigma_2 \ldots$ is an ω-word over σ and τ is a *time sequence* - a strictly monotonic sequence of non-negative reals. We require the time sequence to be *non-Zeno*, i.e. unbounded. We also talk of the word as being constructed from *timed events* (σ_i, τ_i).

Models of MTL formulae are timed words $\rho = (\sigma, \tau)$ over the alphabet AP of atomic propositions. Satisfaction of a formula is defined only at positions i in the timed word rather than at every time point; we say that the timed word ρ satisfies φ if $(\rho, 1) \models \varphi$, define $L(\varphi)$ to be the set of timed words which satisfy φ and say that φ is satisfiable if $L(\varphi) \neq \emptyset$.

We say that a formula in which negation is applied only to atomic propositions is in *positive normal form* and recall that any formula of MTL can be rewritten as an equivalent formula in positive normal form. Using this characterisation, we now define a restricted fragment of MTL.

Definition 5.18. A positive normal form MTL formula is contained in the fragment *Safety* MTL if the interval I in each occurrence of the until modality $\mathbf{U}_I$ has finite length.

Note that this Safety fragment places no restriction on the next modality $\bigcirc_I$ or the dual until modality $\widetilde{\mathbf{U}}_I$.

Example. The types of properties expressible in Safety MTL include *invariance*, which may be expressed by the formula $\Box\varphi$ that requires that φ always holds, and *bounded response*, which may be expressed by the formula $\Box(\varphi \rightarrow \Diamond_{\leq 5}\psi)$ that requires that each time φ holds, ψ holds at most 5 time units later.

5.6.2 Encoding Fair Termination

Given an insertion channel machine $\mathcal{C} = (S, s_0, \Sigma, \Delta)$, we wish to construct a Safety MTL formula $\varphi_{\mathcal{C}}$ such that $\varphi_{\mathcal{C}}$ is satisfiable if and only if $\mathcal{C}$ has an infinite fair run starting from an empty initial channel. In light of the construction in Section 5.4.1, we assume without loss of generality that $\mathcal{C}$ has a distinguished *reliable* symbol $\triangleright$ which does not suffer from insertion errors and $\mathcal{C}$ maintains a single copy of $\triangleright$ on the channel at all times. This feature allows $\mathcal{C}$ to use only operations from Op (i.e. no renaming or occurrence testing operations) while retaining all the power of an insertion channel machine with renaming. The key idea of this construction is that we can capture the channel discipline with a formula such as $\Box(a! \rightarrow \Diamond_{=1} a?)$ which ensures that every $a!$ event is followed exactly one time unit later by an $a?$ event. Note that this formula does not require that every $a?$ event is preceded by an $a!$ event (see Section 2.5.3 for details) so we can only encode an insertion channel machine this way.

Let $\delta(s, op) \stackrel{\text{def}}{=} \{t \mid (s, op, t) \in \Delta\}$ for each $s \in S$, $op \in Op$ and let $Op' = Op \setminus \{\triangleright?, \triangleright!\}$. As the set of events, we choose $S \times Op$ and assume that at most one event occurs at each time instant (so time is strictly monotonic).

We then construct a number of Safety MTL formulae to simulate $\mathcal{C}$ as follows:

$$\begin{aligned}
\varphi_{init} &= (s_0, \triangleright!) \\
\varphi_{trans} &= \bigwedge_{s\in S} \bigwedge_{o\in Op'} \Box \left[(s,o) \to \bigcirc \left(\bigvee_{t\in\delta(s,o)} \bigvee_{u\in Op} (t,u)\right)\right] \\
\varphi_{fair} &= \bigwedge_{s\in S} \bigwedge_{\sigma\in\Sigma} \Box \left[(s,\sigma!) \to \Diamond_{=1} \left(\bigvee_{t\in S} (t,\sigma?)\right)\right] \\
\varphi_{marker} &= \begin{array}{l} \bigwedge_{s\in S} \Box \left[(s,\triangleright!) \to \Box_{<1} \bigwedge_{t\in S} \left(\neg(t,\triangleright?) \wedge \neg(t,\triangleright!)\right)\right] \\ \bigwedge_{s\in S} \Box \left[(s,\triangleright?) \to \bigcirc \left(\bigvee_{t\in\delta(s,\triangleright?)} (t,\triangleright!)\right)\right] \end{array}
\end{aligned}$$

The simulation encoded by these formulae uses one time unit for each cycle of the channel, crucially exploiting the density of time to allow an unbounded channel. Each event contains an operation and corresponding state - this is the state that the channel machine occupies immediately before the operation, hence the formula φ_{init} ensures that the simulation begins with the channel machine in its initial state s_0 and that its first operation writes the special $\triangleright$ symbol to the channel. The transition relation of the channel machine specifies which states may result from the execution of a particular operation in a given state and these results are checked by φ_{trans}, which ensures that the next operation is started in a correct state.

Since the channel machine may perform as many read operations as it likes (consuming symbols placed on the channel by insertion errors), the only restriction we make on these transitions is a *fairness* constraint, which requires that each write operation be followed one cycle later by a corresponding read operation. In our simulation, this corresponds to following an event which contains a write operation with one containing a corresponding read operation one time unit later, and is ensured by φ_{fair}.

We now only need consider the end marker $\triangleright$ and note that we require three additional properties of this symbol. Firstly, it must be written immediately after being read. Secondly, only one copy may be on the channel at a time, hence it cannot be written during the cycle (time unit) following each $\triangleright!$ (and preceding the next $\triangleright?$). Thirdly, $\triangleright$ is never an

insertion error, so it cannot be read until a complete cycle (time unit in the simulation) has passed since the last write. The formula φ_{marker} ensures that these additional properties are met.

By letting $\varphi_{\mathcal{C}} = \varphi_{init} \wedge \varphi_{trans} \wedge \varphi_{fair} \wedge \varphi_{marker}$, we obtain the following proposition.

Proposition 5.19. *$\varphi_{\mathcal{C}}$ is satisfiable if and only if the channel machine $\mathcal{C}$ has an infinite fair run starting from an empty initial channel.*

Proof. It is clear that an infinite timed word ρ which satisfies $\varphi_{\mathcal{C}}$ represents an infinite sequence of transitions of the channel machine, which must include an infinite number of transitions with operand $\triangleright!$ by φ_{marker}. Hence this corresponds to a run with infinitely many cycles and is therefore fair.

Moreover, any infinite fair run of $\mathcal{C}$ will contain infinitely many cycles by our requirement on $\triangleright$. Hence it is easy to see that we could add times to the transitions in such a run in order to produce a timed word ρ which would satisfy $\varphi_{\mathcal{C}}$ by placing writes and their corresponding reads at distance 1. □

Note that the size of $\varphi_{\mathcal{C}}$ is polynomial in the size of $\mathcal{C}$ since φ_{init} has constant size, φ_{trans} is $O(|S|^2|Op|^2)$, φ_{fair} is $O(|S|^2|\Sigma|)$ and φ_{marker} is $O(|S|^2)$. Moreover, we can construct $\varphi_{\mathcal{C}}$ in space logarithmic in $|\mathcal{C}|$ by using four pointers to elements of $\mathcal{C}$, each requiring logarithmic space, to perform the required iterations.

Note that for any insertion channel machine with renaming $\mathcal{C}$, the construction of Section 5.4.1 produces an insertion channel machine $\mathcal{D}$ with only a reliable end marker whose size is exponential in the size of $\mathcal{C}$ and requires space linear in the size of $\mathcal{C}$ to perform.

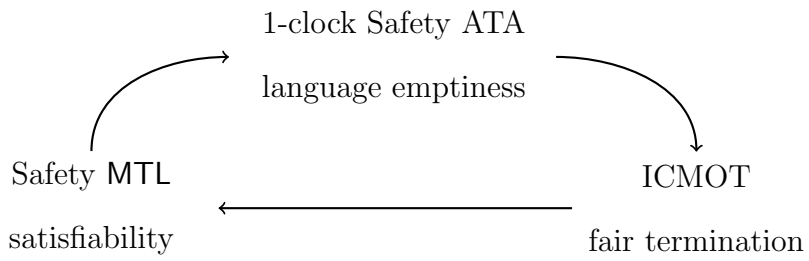
However, the channel machine constructed by Theorem 5.17 uses no renamings and only uses occurrence tests to verify the reliability of the $\triangleright$ symbol. We can therefore apply Proposition 5.19 to such a channel machine (which has a $\textsc{Space}(F_4(n))$-hard fair

termination problem) in order to conclude the following.

Theorem 5.20. *The Safety* MTL *satisfiability problem is* SPACE$(F_4(n))$*-hard.*

5.7 Safety Alternating Timed Automata

Having established a connection between insertion channel machines with renaming and Safety MTL, we now illuminate the relationship between both of these formalisms and that of alternating timed automata. In this section we consider 1-clock *safety* alternating timed automata, which can be thought of as having only accepting states. We first give a definition of these alternating timed automata following the acceptance game semantics considered elsewhere in this dissertation and in [LW05]. In Section 5.7.1, we demonstrate that Safety MTL satisfiability reduces to the language emptiness problem for these safety alternating timed automata by adapting a construction from [OW05] to this semantics. Finally, in Section 5.7.2, we show that this language emptiness problem itself reduces to the fair termination problem for insertion channel machines with renaming, completing the cycle of relationships seen in the introduction:



We recall the definition of alternating timed automata from Section 2.4.

Definition 2.9. *An* alternating timed automaton *is a tuple* $\mathcal{A} = (S, s_0, \Sigma, C, F, \delta)$ *where* S *is a set of locations with* $s_0 \in S$ *the initial location and* $F \subseteq S$ *the subset of accepting locations. It consumes letters from the alphabet* Σ*, has a set* C *of clock variables and the transition function* $\delta : S \times \Sigma \times \Phi(C) \to \mathcal{B}^+(Moves(C, S))$.

Recall also that we assume that for a given location and letter, the transitions of such an automaton have disjoint constraints over the clock variables. We define acceptance of a timed word w by such an automaton in terms of a game $G(\mathcal{A}, w)$ which has one round for each event in w and whose positions record the current location and clock valuation of the automaton as well as the round number.

In a round starting in position (s_i, ν_i, i), the Automaton player first chooses a set of moves which agree with the transition function based on s_i, ν_i and the event $(\sigma_{i+1}, \tau_{i+1})$ and loses if he cannot choose such a set (because there is no transition $\delta(s_i, \sigma_{i+1}, \varphi)$ such that $\nu_i + (\tau_{i+1} - \tau_i) \in [\varphi]$). The Pathfinder player then responds by choosing a single move from this set.

An infinite play is won by Automaton if and only if there exists an accepting location which occurs infinitely often in the play; a timed word w is then accepted by $\mathcal{A}$ if Automaton has a winning strategy in $G(\mathcal{A}, w)$.

We now define the restricted class of safety alternating timed automata:

Definition 5.21. A *safety* alternating timed automaton is one in which all locations are accepting.

Note that this means that any infinite play of the acceptance game is won by the Automaton player, so the problem of whether he has a strategy to produce one is intuitively closely linked to the fair termination problem for insertion channel machines. In the following, we focus on *1-clock* safety alternating timed automata which have only the single clock variable x.

This restriction intuitively captures the semantic definition of safety properties outlined in Section 5.6 since any play of the acceptance game $G(\mathcal{A}, w)$ which is won by Pathfinder only has a finite number of rounds and thus only depends on a finite bad prefix w_i of w.

5.7.1 Embedding Safety MTL

Given a Safety MTL formula φ in positive normal form, we seek to define a safety alternating timed automaton $\mathcal{A}_\varphi$ with only a single clock variable x such that $L(\mathcal{A}_\varphi) = L(\varphi)$. Since alternating timed automata are closed under union and intersection, it is clear how to define $\mathcal{A}_\varphi$ in case φ is an atomic formula or the negation of an atomic formula. We thus consider without loss of generality only the cases when the outermost connective in φ is $\mathbf{U}_I$, $\widetilde{\mathbf{U}}_I$ or $\bigcirc_I$ and adapt the construction of [OW06b] to the acceptance game semantics.

Define the closure of φ, denoted $cl(\varphi)$ to consist of all subformulae of φ whose outermost connective is $\mathbf{U}_I$, $\widetilde{\mathbf{U}}_I$ or $\bigcirc_I$. We define $\mathcal{A}_\varphi = (S, \varphi^i, \Sigma, \{x\}, \delta)$ as follows. The set of locations is $S = cl(\varphi) \cup \{\varphi^i, T\}$. We call φ^i the *initial copy* of φ; it is the initial location of $\mathcal{A}_\varphi$. We use the location T as a sink location — loops on every event are always available at T, so if T is reached in a play of the acceptance game, Automaton will be able to win that play.

We define the transition function δ so that the presence of the position $(\psi, \mathbf{0}, i+1)$ (where the location is ψ, the clock value is 0 and the round number is $i+1$) during a play of $G(\mathcal{A}_\varphi, \omega)$ ensures that the $(\omega, i) \models \psi$, i.e. that the input word satisfies ψ from the current letter. To enforce this requirement, when ψ is encountered the automaton starts a fresh clock and thereafter propagates ψ from position to position in the game until all the obligations that it stipulates are discharged. We first define an auxiliary function $init(\psi, a)$ for each subformula ψ of φ and $a \in \Sigma$ as follows:

$$
\begin{aligned}
init(\psi_1 \ \mathbf{U}_I \ \psi_2, a) &= (\{x\}, \psi_1 \ \mathbf{U}_I \ \psi_2) \wedge init(\psi_1, a) \\
init(\psi_1 \ \widetilde{\mathbf{U}}_I \ \psi_2, a) &= (\{x\}, \psi_1 \ \widetilde{\mathbf{U}}_I \ \psi_2) \\
init(\bigcirc_I \psi_1, a) &= (\{x\}, \bigcirc_I \psi_1) \\
init(\psi_1 \wedge \psi_2, a) &= init(\psi_1, a) \wedge init(\psi_2, a) \\
init(\psi_1 \vee \psi_2, a) &= init(\psi_1, a) \vee init(\psi_2, a)
\end{aligned}
$$

$$init(b,a) = \begin{cases} (\{x\}, T) & \text{if } a = b \\ \emptyset & \text{if } a \neq b \end{cases} \quad \text{for } b \in \Sigma$$

$$init(\neg b,a) = \begin{cases} \emptyset & \text{if } a = b \\ (\{x\}, T) & \text{if } a \neq b \end{cases} \quad \text{for } b \in \Sigma$$

Then δ is defined by:

$$\begin{aligned}
\delta(T, a, x \geq 0) &= (\emptyset, T) \\
\delta(\varphi^i, a, x \geq 0) &= init(\varphi, a) \\
\delta(\bigcirc_I \psi_1, a, x \in I) &= init(\psi_1, a) \\
\delta(\psi_1 \ \widetilde{\mathbf{U}}_I \ \psi_2, a, x \in I) &= init(\psi_2, a) \wedge \Big(init(\psi_1, a) \vee (\emptyset, \psi_1 \ \widetilde{\mathbf{U}}_I \ \psi_2) \Big) \\
\delta(\psi_1 \ \widetilde{\mathbf{U}}_I \ \psi_2, a, x \notin I) &= init(\psi_1, a) \vee (\emptyset, \psi_1 \ \widetilde{\mathbf{U}}_I \ \psi_2) \\
\delta(\psi_1 \ \mathbf{U}_I \ \psi_2, a, x \in I) &= init(\psi_2, a) \vee (init(\psi_1, a) \wedge (\emptyset, \psi_1 \ \mathbf{U}_I \ \psi_2)) \\
\delta(\psi_1 \ \mathbf{U}_I \ \psi_2, a, x < I) &= init(\psi_1, a) \wedge (\emptyset, \psi_1 \ \mathbf{U}_I \ \psi_2)
\end{aligned}$$

where $x < I$ means that for all $t \in I$, $x < t$.

Note that this definition adheres to the *locality* condition prescribed in [OW06b] which requires the automaton to reset the single clock x whenever the location changes.

Proposition 5.22. $L(\mathcal{A}_\varphi) = L(\varphi)$

Proof Sketch. We first show that $L(\mathcal{A}_\varphi) \subseteq L(\varphi)$. To this end, let $\rho = (\sigma, \tau)$ be an infinite timed word in $L(\mathcal{A}_\varphi)$. Write $d_i = \tau_{i+1} - \tau_i$ and suppose that Automaton has a winning strategy in $G(\mathcal{A}_\varphi, \rho)$.

Suppose that $(M_1, m_1), \ldots, (M_i, m_i)$ is a partial play of $G(\mathcal{A}_\varphi, \rho)$ where the models M_j are chosen by Automaton according to this strategy (and $m_j \in M_j$ for each j). We claim that for each subformula ψ of φ and each i, if $M_i \models init(\psi, \sigma_i)$ then $(\rho, i) \models \psi$. We prove this claim by structural induction on ψ. The only non-trivial cases are when the outermost

connective of ψ is a temporal modality. We consider the case $\psi \stackrel{\text{def}}{=} \psi_1 \ \mathbf{U}_I \ \psi_2$ by way of example.

Suppose that $\psi \stackrel{\text{def}}{=} \psi_1 \ \mathbf{U}_I \ \psi_2$ and $M_i \models \mathit{init}(\psi, \sigma_i)$. Then $\mathit{init}(\psi, \sigma_i) = (\{x\}, \psi) \wedge \mathit{init}(\psi_1, \sigma_i)$ and there exists a Pathfinder move m_i which moves the game to the state $(\psi, 0, i+1)$. Analysing the definition of the transition function δ we can show that there exists a value of k and sequence of moves of the game $(M_i, m_i), \ldots, (M_k, m_k)$ for which for each $i \leq j < k$, $M_j \models \mathit{init}(\psi_1, \sigma_j) \wedge (\emptyset, \psi_1 \ \mathbf{U}_I \ \psi_2)$, $m_j = (\emptyset, \psi_1 \ \mathbf{U}_I \ \psi_2)$ and $M_k \models \mathit{init}(\psi_2, \sigma_k)$ with $\tau_k - \tau_i \in I$. Such a k must exist as Automaton has a winning strategy and this is the only transition which leaves the location ψ or resets the clock x. Time diverges since ρ is infinite and there are no transitions available from location l when x becomes too large. From the induction hypothesis it is clear that this implies that $(\rho, i) \models \psi$.

Having proved the claim we observe that $(\varphi, 0) \in M_1$, and so $M_1 \models \mathit{init}(\varphi, \sigma_1)$. Thus, applying the claim in the case $i = 1$ and $\psi \equiv \varphi$, we immediately get that $\rho \models \varphi$ whenever Automaton has a winning strategy in $G(\mathcal{A}_\varphi, \rho)$. This completes the proof that $L(\mathcal{A}_\varphi) \subseteq L(\varphi)$.

It remains to show the converse inclusion: $L(\varphi) \subseteq L(\mathcal{A}_\varphi)$. The extraction of a winning strategy for Automaton follows easily from the definition of satisfaction for φ and the transition relation for $\mathcal{A}_\varphi$. □

Observe that since the size of $cl(\varphi)$ is linear in the size of φ, the size of S is also. δ has at most two transitions for each element of $S \times \Sigma$ and each such transition has size at most linear in the size of S. It can therefore easily be seen that a fixed number of pointers to sections of φ or Σ suffice to construct $\mathcal{A}_\varphi$. We therefore conclude that

Proposition 5.23. *Given an alphabet Σ and formula φ of Safety* MTL *over Σ, there exists an algorithm which constructs a 1-clock safety alternating timed automaton $\mathcal{A}_\varphi$ in space logarithmic in $|\Sigma|$ and $|\varphi|$ such that $L(\mathcal{A}_\varphi) = L(\varphi)$.*

5.7.2 Closing the Loop

In this section we reduce the language emptiness problem for a 1-clock safety alternating timed automaton $\mathcal{A} = (S, s_0, \Sigma, \{x\}, \delta)$ to the fair termination problem for an insertion channel machine with renaming, denoted $\mathcal{C}_\mathcal{A}$.

This construction is similar to the time-abstract bisimulation quotient method of [OW05] which yielded a well-structured transition system. It can also be considered as extending the translation of alternating timed automata to channel machines with renaming presented in [BMOW07] to infinite timed words. Before embarking on the construction, we make a number of preliminary definitions.

For a strategy $\mathcal{S}$, write $Lev_i(\mathcal{S}, w)$ for the set of positions which the acceptance game $G(\mathcal{A}, w)$ may occupy at the end of the ith round if Automaton follows the strategy $\mathcal{S}$ (a *level* of the corresponding game tree). Note that the choices available to Automaton and Pathfinder in the first i rounds of $G(\mathcal{A}, w)$ are the same as those available in $G(\mathcal{A}, v)$ for any timed word v with $v_j = w_j$ for all $1 \leq j \leq i$, hence $Lev_i(\mathcal{S}, w) = Lev_i(\mathcal{S}, v)$ for such a v. We intend our channel machine $\mathcal{C}_\mathcal{A}$ to simulate the entirety of $Lev_i(\mathcal{S}, w)$ in a single configuration for each i in turn.

If N is the largest constant appearing in $\mathcal{A}$, we define the set $\mathbb{C}_\mathcal{A} = \{0, 1, \ldots, N+1\}$ and the set of clock regions $Reg_\mathcal{A} = \{0, 0^+, 1, 1^+, \ldots, N, \bot\}$. The region n for $0 \leq n \leq N$ represents the clock valuation ν with $\nu(x) = n$. The region n^+ for $0 \leq n < N$ represents the set of valuations $\{\nu \mid n < \nu(x) < n+1\}$. The region $\bot$ represents all valuations ν where $N < \nu(x)$. For a clock constraint φ and region r, we say that $r \in [\varphi]$ if and only if for all valuations ν represented by r, $\nu \in [\varphi]$. Note that by our choice of regions and the Disjointness condition on the transition function of $\mathcal{A}$, for each location s, letter σ and region r, there is at most one φ such that $\delta(s, \sigma, \varphi)$ is defined and $r \in [\varphi]$.

An *abstract position* of the acceptance game is then an element from $S \times Reg_\mathcal{A}$, i.e. a location of $\mathcal{A}$ along with a clock region. We say that the abstract position (s, r) represents

the position (s', ν, i) if and only if $s = s'$ and r represents ν. We define a *time-successor* operation *Succ* on abstract positions by

$$Succ(s,t) = \begin{cases} (s, t^+) & \text{if } t < N \\ (s, \bot) & \text{if } t \in \{N, \bot\} \end{cases}$$

$$Succ(s, t^+) = (s, t+1)$$

We then extend this operation pointwise to sets of abstract positions by

$$Succ(\{(s_1,t_1),\ldots,(s_r,t_r)\}) = \{Succ(s_1,t_1),\ldots,Succ(s_r,t_r)\}\,.$$

We are now in a position to define the channel machine $\mathcal{C}_\mathcal{A}$ which will simulate the 1-clock safety alternating timed automaton $\mathcal{A}$.

As mentioned above, we wish each run of $\mathcal{C}_\mathcal{A}$ to simulate all possible evolutions of an acceptance game for $\mathcal{A}$ where the Automaton's strategy is fixed. We further wish to ensure that if such a run is infinite and fair then the choices of Automaton form a winning strategy for this acceptance game (and hence that there exists some word in $L(\mathcal{A})$). This means that we wish every infinite fair run to simulate infinitely many rounds of the acceptance game and correspond to a non-Zeno play.

$\mathcal{C}_\mathcal{A}$ will separately simulate the advance of time and the discrete transition chosen in each round of $\mathcal{A}$'s acceptance game, with discrete transitions corresponding to renaming operations and the advance of time corresponding to transitions which read from and write to the channel. We wish to force an infinite fair play to interleave infinitely many transitions of each type, so we use a $\triangleright$ symbol to ensure that infinitely many cycles occur and record the number of cycles since the last renaming occurred in the state, preventing $\triangleright$ from being read when it would increase this number above $N+1$. To enforce the interleaving, we record a location from $L = \{c, d, e, f\}$ in the channel machine's state where c indicates that a time-advance transition was performed last and d indicates that a renaming transition was

performed last. e and f are used to enforce writing $\triangleright$ after reading it while remembering whether c or d occurred last.

These three restrictions on transitions of $\mathcal{C}_{\mathcal{A}}$ ensure that every infinite fair run simulates an infinite sequence of rounds of $\mathcal{A}$'s acceptance game which have timestamps that are non-Zeno and strictly increasing.

Each channel letter is either a set of abstract positions, from the alphabet $\mathcal{P}(S \times Reg_{\mathcal{A}})$, or the special symbol $\triangleright$. Our intent is that all abstract positions in the same letter have equal fractional parts of their clock valuation and the fractional parts descend along the channel (so those abstract positions in the letter at the head of the channel will be first to change when time advances). We will separately record those abstract positions with integer valuations in the state of $\mathcal{C}_{\mathcal{A}}$. This is the last component of the states of our channel machine, which are drawn from the set $L \times \mathbb{C}_{\mathcal{A}} \times \mathcal{P}(S \times Reg_{\mathcal{A}})$.

Example. Suppose that the largest constant in $\mathcal{A}$ is 3 and the Automaton player uses strategy $\mathcal{S}$ in the acceptance game $G(\mathcal{A}, w)$. Suppose further that the possible positions of this game at the end of the ith round are

$$Lev_i(\mathcal{S}, w) = \{(s_1, 2.7, i), (s_2, 4.0, i), (s_3, 5.5, i), (s_4, 3.0, i), (s_5, 1.7, i)\} .$$

Partitioning this set by equality of fractional parts and listing in descending order, we obtain the sets $\{(s_2, 4.0, i), (s_4, 3.0, i)\}$, $\{(s_1, 2.7, i), (s_5, 1.7, i)\}$ and $\{(s_3, 5.5, i)\}$. We replace each clock valuation with its corresponding region and delete the round numbers to obtain the sets of abstract positions $\{(s_2, \perp), (s_4, 3)\}$, $\{(s_1, 2^+), (s_5, 1^+)\}$ and $\{(s_3, \perp)\}$ (which retain the ordering of the fractional parts).

The channel machine stores the set $\{(s_2, \perp), (s_4, 3)\}$ with integer valuations in its state, while the channel stores the two letters

$\{(s_1, 2^+), (s_5, 1^+)\}$	$\{(s_3, \perp)\}$

We now define the transition rules Δ of $\mathcal{C}_{\mathcal{A}}$. There are three types of time-advancing transitions, which use the time-successor relation *Succ* on abstract positions defined above. However, these transitions do not precisely correspond to *Succ* since time advances need not alter the n^+ regions (as for every valuation ν in n^+ there exists $t \in \mathbb{R}_+$ such that $\nu + t \in n^+$).

Time advances which do not increase the region of any clocks are handled by transitions of the form

$$((d, j, \emptyset), nop, (c, j, \emptyset)) \in \Delta$$

for $j \in \mathbb{C}_{\mathcal{A}}$.

Time advances which increase integer clocks just above their integer value (without affecting other clock regions) are handled by transitions of the form

$$((l, j, \alpha), Succ(\alpha)!, (c, j, \emptyset)) \in \Delta$$

for $l \in \{c, d\}$, $j \in \mathbb{C}_{\mathcal{A}}$ and any $\alpha \in \mathcal{P}(S \times Reg_{\mathcal{A}})$.

Time advances which increase those non-integer clocks with the highest fractional part to the next integer value (without affecting other clock regions) are handled by transitions of the form

$$((l, j, \emptyset), \alpha?, (c, j, Succ(\alpha))) \in \Delta$$

for $l \in \{c, d\}$, $j \in \mathbb{C}_{\mathcal{A}}$ and any $\alpha \in \mathcal{P}(S \times Reg_{\mathcal{A}})$.

Discrete transitions of $\mathcal{A}$ are simulated by renaming operations. These are described by a letter $\sigma \in \Sigma$ and a *reset set* of automaton locations $Z \subseteq S$ which correspond to those successor positions whose clock is reset by this transition. We then define a relation between abstract positions and channel letters $\widetilde{R}^Z_\sigma \subseteq (S \times Reg_{\mathcal{A}}) \times \mathcal{P}(S \times Reg_{\mathcal{A}})$ by

$$(s, r)\ \widetilde{R}^Z_\sigma\ (Q \times \{r\}) \text{ if and only if } (\{\emptyset\} \times Q) \cup (\{\{x\}\} \times Z) \models \delta(s, \sigma, \varphi) \text{ and } r \in [\varphi]\,.$$

We lift this relation to a renaming R^Z_σ of channel letters by setting $\triangleright\ R^Z_\sigma\ \triangleright$ and

$$\text{for } \alpha, \beta \in \mathcal{P}(S \times Reg_{\mathcal{A}}), \alpha\ R^Z_\sigma\ \beta \text{ if and only if for all } (s, r) \in \alpha, (s, r)\ \widetilde{R}^Z_\sigma\ \beta\,.$$

The intent of these transitions is that the renaming takes care of updating the letters of the channel in place, while the set Z describes which abstract positions must be added to those stored in the control state of the channel machine. We hence have transitions

$$((c, j, \alpha), R_\sigma^Z, (d, 0, \beta)) \in \Delta$$

for all $j \in \mathbb{C}_\mathcal{A}$ and R_σ^Z where $\beta = \alpha' \cup (Z \times \{0\})$ and $\alpha \; R_\sigma^Z \; \alpha'$.

Example. Suppose that $\mathcal{A}$ contains the transitions

$$\delta(s, \sigma, 3 < x < 4) = ((\emptyset, t) \wedge (\emptyset, s)),$$

$$\delta(t, \sigma, 3 < x < 4) = ((\emptyset, t) \vee (\emptyset, u)) \text{ and}$$

$$\delta(u, \sigma, 3 < x < 4) = (\emptyset, u) \wedge (\{x\}, s).$$

The first of these transitions moves from location s with $3 < \nu(x) < 4$ to either location s or t (Pathfinder's choice) with ν unchanged. The second transition moves from location t with $3 < \nu(x) < 4$ to either location t or u (Automaton's choice) with ν unchanged. The third transition moves from u to Pathfinder's choice of location u with ν unchanged or s with valuation $\mathbf{0}$ (where $\mathbf{0}(x) = 0$).

These transitions hence induce a relation $\widetilde{R}_\sigma^\emptyset$ such that $(s, 3^+) \; \widetilde{R}_\sigma^\emptyset \; \{(s, 3^+), (t, 3^+)\}$, $(t, 3^+) \; \widetilde{R}_\sigma^\emptyset \; \{(t, 3^+)\}$ and $(t, 3^+) \; \widetilde{R}_\sigma^\emptyset \; \{(u, 3^+)\}$. The abstract position $(s, 3^+)$ is related to a set of successor positions since Pathfinder may choose between them, while the abstract position $(t, 3^+)$ is separately related to two successor positions since Automaton may choose between them. Note that u is not related to any letter by $\widetilde{R}_\sigma^\emptyset$ since the Pathfinder may force the game into a position with location s and the clock reset, yet s is not contained in our reset set $\emptyset$.

These transitions also induce a relation $\widetilde{R}_\sigma^{\{s\}}$ — note that if an abstract position (s, r) is related to a channel letter α by $\widetilde{R}_\sigma^\emptyset$, then also $(s, r) \; \widetilde{R}_\sigma^{\{s\}} \; \alpha$ since $\emptyset \subseteq \{s\}$. Moreover, we have $(u, 3^+) \; \widetilde{R}_\sigma^{\{s\}} \; \{(u, 3+)\}$ as we may transition from location u if we allow Pathfinder to choose the successor abstract position $(s, 0)$.

The rules in Δ which correspond to these relations will then induce many possible transitions of channel machine configurations, including the following examples. Note that in each such example, the configuration before the transition has a single letter stored on its channel.

$$((c, j, \emptyset), \{(s, 3^+)\}) \xrightarrow{R^{\emptyset}_{\sigma}} ((d, 0, \emptyset), \{(s, 3^+), (t, 3^+)\})$$

$$((c, j, \emptyset), \{(s, 3^+), (t, 3^+)\}) \xrightarrow{R^{\emptyset}_{\sigma}} ((d, 0, \emptyset), \{(s, 3^+), (t, 3^+)\})$$

$$((c, j, \emptyset), \{(s, 3^+), (t, 3^+)\}) \xrightarrow{R^{\emptyset}_{\sigma}} ((d, 0, \emptyset), \{(s, 3^+), (t, 3^+), (u, 3^+)\})$$

$$((c, j, \emptyset), \{(s, 3^+), (t, 3^+), (u, 3^+)\}) \xrightarrow{R^{\{s\}}_{\sigma}} ((d, 0, \{(s, 0)\}), \{(s, 3^+), (t, 3^+), (u, 3^+)\})$$

We finally add a number of housekeeping transitions to $\mathcal{C}_{\mathcal{A}}$ which allow the $\triangleright$ symbol to be cycled only when at most $N + 1$ cycles have passed since the last renaming, as follows:

$$((c, j, \alpha), \triangleright?, (e, j + 1, \alpha)) \in \Delta, \qquad ((e, j, \alpha), \triangleright!, (c, j, \alpha)) \in \Delta,$$
$$((d, j, \alpha), \triangleright?, (f, j + 1, \alpha)) \in \Delta \quad \text{and} \quad ((f, j, \alpha), \triangleright!, (d, j, \alpha)) \in \Delta$$

for $j, j + 1 \in \mathbb{C}_{\mathcal{A}}$ and any $\alpha \in \mathcal{P}(S \times \mathit{Reg}_{\mathcal{A}})$.

This completes the construction of $\mathcal{C}_{\mathcal{A}}$, which we summarise as follows:

$\mathcal{C}_{\mathcal{A}}$ has set of states $L \times \mathbb{C}_{\mathcal{A}} \times \mathcal{P}(S \times \mathit{Reg}_{\mathcal{A}})$, initial state $(e, 0, \{(s_0, 0)\})$, channel alphabet $\mathcal{P}(S \times \mathit{Reg}_{\mathcal{A}}) \cup \{\triangleright\}$ and transition relation Δ as defined above.

The construction of $\mathcal{C}_{\mathcal{A}}$ requires space exponential in the size of $\mathcal{A}$, since it must enumerate through $\mathcal{P}(S \times \mathit{Reg}_{\mathcal{A}})$ and space exponential in the size of $\mathcal{A}$ is required to represent each element. This is due to the fact that $\mathit{Reg}_{\mathcal{A}}$ itself is exponential in the size of $\mathcal{A}$ if the constants in $\mathcal{A}$ are encoded in binary. This leads to the size of $\mathcal{C}_{\mathcal{A}}$ being doubly-exponential in the size of $\mathcal{A}$.

Note, however, that this definition is resistant to insertion errors since simulating additional positions of $\mathcal{A}$'s acceptance game cannot generate an infinite fair run of $\mathcal{C}_{\mathcal{A}}$ where no error-free infinite fair run existed. To avoid the exponential blow-up inherent in using the channel alphabet $\mathcal{P}(S \times \mathit{Reg}_{\mathcal{A}})$, one might hope to represent sets of abstract positions on the

channel using demarcation symbols { and }. The set $\{(s,r),(s',r')\}$ would then be stored as the sequence of symbols $\boxed{\{}\boxed{(s,r)}\boxed{(s',r')}\boxed{\}}$ instead of the single symbol $\boxed{\{(s,r),(s',r')\}}$ used by $\mathcal{C}_\mathcal{A}$. Whilst this encoding might be exponentially more compact, it would be vulnerable to the insertion of additional pairs of } and { symbols which would split the sets of abstract positions and break the simulation invariant that these sets simulate a partition of the positions of some $Lev_i(\mathcal{S}, w)$ by equality of fractional parts of clock valuations.

We assert the correctness of the construction of $\mathcal{C}_\mathcal{A}$ with the following proposition, which is proved in Appendix B.

Proposition 5.24. *$L(\mathcal{A}) \neq \emptyset$ if and only if $\mathcal{C}_\mathcal{A}$ has an infinite fair run.*

The more difficult direction of this proof is extracting a timed word in $L(\mathcal{A})$ from an infinite fair run of $\mathcal{C}_\mathcal{A}$. Whilst the sequence of letters σ_i in such a word can be taken from the sequence of renamings $R_{\sigma_i}^{Z_i}$ which occur in the run, the extraction of corresponding timestamps requires a more subtle argument which relies on the density of time to obtain delays that do not advance any simulated clock regions. The key intuition of this proof is that each cycle of the channel should correspond to one time unit in the simulated timed word.

We can thus conclude the following:

Proposition 5.25. *For each 1-clock safety alternating timed automaton $\mathcal{A}$, we can construct an insertion channel machine with renaming $\mathcal{C}_\mathcal{A}$ in space exponential in $|\mathcal{A}|$ such that there exists an infinite fair computation of $\mathcal{C}_\mathcal{A}$ if and only if $L(\mathcal{A}) \neq \emptyset$.*

For any 1-clock safety alternating timed automata $\mathcal{A}$, we can first apply Proposition 5.25 to obtain the insertion channel machine with renaming $\mathcal{C}_\mathcal{A}$ and secondly apply the decision procedure of Theorem 5.16 to decide whether $\mathcal{C}_\mathcal{A}$ has an infinite fair computation and hence whether $L(\mathcal{A})$ is nonempty. We thus conclude

Theorem 5.26. *The language emptiness problem is decidable for 1-clock safety alternating timed automata.*

We can then combine this result with that of the previous section to obtain:

Proposition 5.27. *For any Safety* MTL *formula φ, we can compute an insertion channel machine with renaming $\mathcal{C}_\varphi$ in space exponential in $|\varphi|$ such that φ is satisfiable if and only if $\mathcal{C}_\varphi$ has an infinite fair computation.*

Proof. Using the construction in Section 5.7.1, we first compute an automaton $\mathcal{A}_\varphi$ with $L(\varphi) = L(\mathcal{A}_\varphi)$. We then apply Proposition 5.25 to this automaton to obtain an insertion channel machine $\mathcal{C}_\varphi$ such that there exists an infinite fair computation of $\mathcal{C}_\varphi$ if and only if $L(\mathcal{A}_\varphi) \neq \emptyset$. As φ is satisfiable exactly if $L(\varphi) = L(\mathcal{A}_\varphi) \neq \emptyset$, this completes the proof. □

We can similarly combine Proposition 5.27 with Theorem 5.16 to conclude the following result, first proved in [OW06b].

Theorem 5.28. *The Safety* MTL *satisfiability problem is decidable.*

5.8 Summary

In this chapter we have investigated the complexity of the fair termination problem for different types of insertion channel machines. In the case that these machines have only read and write operations available, we showed that the problem is PSPACE-complete.

When considering machines with more advanced operations, we have given reductions between machines which employ the facilities of renaming, occurrence testing and a reliable end-marker. We exhibited a decision procedure for the fair termination problem for these machines using Higman's Lemma and further proved that the problem is at least SPACE($F_4(n)$)-hard. These results still leave open the question of the exact complexity of

the fair termination problem, but we believe that with future work a more subtle decision procedure may be found which has F_ω complexity similar to the Ackermann function.

Having obtained these complexity results, we employed these insertion channel machines with renaming as a technical tool for the analysis of a restricted class of alternating timed automata over infinite timed words. We gave a new proof that the language emptiness problem for this class of 1-clock safety alternating timed automata is decidable by reduction to the fair termination problem for insertion channel machines with renaming. We also transfer the $\textsc{Space}(F_4(n))$ hardness result for this problem by reducing the Fair Termination problem to the satisfiability problem for the Safety fragment of the temporal logic MTL and thence to language emptiness for 1-clock safety alternating timed automata.

We note that the translation of 1-clock alternating timed automata to insertion channel machines with renaming could be adapted to the case where the automata have a Büchi acceptance condition (and not all locations are accepting), but then the channel machine problem corresponding to language emptiness of these automata would be the *recurrent state problem*; this problem was shown to be undecidable in [OW06a], so the restriction to the class of 1-clock safety alternating timed automata in this chapter is crucial.

Chapter 6

Discussion

6.1 Conclusions

In this dissertation we sought restricted classes of alternating timed automata whose language emptiness problem is decidable. We exhibited two such classes and considered real-time extensions of Church's synthesis problem under one of our restrictions.

In Chapter 3 we extended the paradigm of *time-bounded verification* to alternating timed automata. We proved that even for alternating timed automata with many clocks, the time-bounded language emptiness problem is decidable. This class complements the previously known class of 1-clock alternating timed automata over finite words since it allows multiple clocks at the cost of being only able to consider executions of fixed duration instead of arbitrary finite duration.

In the course of proving this result, we introduced a bounded real-time extension of McNaughton games. These games are associated with Church's synthesis problem and in Chapter 4, we extended our techniques to prove the decidability of two synthesis problems over the reals under the finite variability interpretation.

We proved that it is decidable whether there exists an $\mathsf{MSO}(<)$-definable causal uni-

formizer for a given $\mathsf{MSO}(<)$ specification over $\langle \mathbb{R}_+, < \rangle$ and exhibited an algorithm which computes such a uniformizer if one exists. We note, however, that there exist $\mathsf{MSO}(<)$ specifications over $\langle \mathbb{R}_+, < \rangle$ for which a causal uniformizer exists but for which there is no $\mathsf{MSO}(<)$-definable causal uniformizer (see Proposition 4.12 for details).

We then applied the techniques seen in Chapter 3 in order to extend this result to the case of bounded intervals of metric reals. We proved that it is decidable whether there exists an $\mathsf{MSO}(<, +1)$-definable causal uniformizer for a given $\mathsf{MSO}(<, +1)$ specification over $\langle [0, N), <, +1 \rangle$ and exhibited an algorithm which computes such a uniformizer if one exists.

In Chapter 5 we considered instead the class of *safety* alternating timed automata, which have only accepting states. We introduced the auxiliary formalism of insertion channel machines with renaming and the safety fragment of the temporal logic MTL. We proved that the three problems of language emptiness for 1-clock safety alternating timed automata, fair termination for insertion channel machines with renaming and satisfiability for Safety MTL are interreducible. Believing that the channel machine formalism offered the best approach to obtaining complexity results for these three problems, we concentrated on this model and proved that the fair termination problem for insertion channel machines with renaming is hard for the complexity class $\textsc{Space}(F_4(n))$ where F_4 is the function at the fourth level of the Grzegorczyk hierarchy [Grz53].

We summarise the hardness results obtained for classes of alternating timed automata in the following table, with grey cells indicating previously known results.

	time-bounded	finite	safety
1-clock	non-elementary	non-primitive recursive	F_4
many clocks	non-elementary	undecidable	undecidable

6.2 Related Work

Timed games are closely related to our work and have been studied in a number of different contexts. There are many examples in the literature and as well as the classical distinctions of turn-based vs concurrent, total vs partial information, type of game arena and requirements on strategies, we identify distinctions of duration and treatment of Zeno plays which are specific to the timed setting.

The acceptance game for alternating timed automata is a turn-based game of total information which usually has infinite duration, but we consider a time-bounded variant in Chapter 3. In either case, the arena is generated from the combination of the input timed word and alternating timed automaton. The only restriction on strategies is that moves occur at points of the input word and this ensures that Zeno plays cannot be generated.

We can think of the causal uniformizers from Chapter 4 as strategies for a type of timed synthesis game of perfect information. In these games, we consider arenas generated by formulae $\varphi(\overline{X}, \overline{Y})$ of $\mathsf{MSO}(<)$ or of $\mathsf{MSO}(<, +1)$. In the former case, we consider games of infinite duration but in the latter case we restrict our attention to time-bounded games. In both cases, the player choosing the $\overline{X}$ variables has only the restriction that his choices must be finitely variable while we require that the player choosing the $\overline{Y}$ variables plays an MSO-definable strategy; this requirement leads to the $\overline{X}$-player being able to choose the time points at which the variables will change and hence to a turn-based game. The $\overline{Y}$-player cannot therefore be responsible for a Zeno play, so our requirement that $\overline{X}$ be finitely variable serves to preclude Zeno plays.

The *realizability* problem weakens the requirements of Church's synthesis problem by removing any restrictions on the strategy of the $\overline{Y}$ player. In the case that φ is a formula of LTL, the realizability problem was shown to be 2-ExpSpace-complete by Pnuelli and Rosner [PR89] compared to a PSpace-complete satisfiability problem [SC85]. In [DGRR09], the authors consider the realizability problem for the decidable ECL fragment of MTL.

Their games become concurrent (the player who chooses the shorter delay determines the time of the next round), have perfect information, infinite duration and place no requirements on the strategies of the players. The authors prove that the realizability problem is undecidable for ECL whether or not Zeno plays are allowed.

The problem of *controller synthesis* takes a different approach by defining a highly asymmetric game. The games employed in this field have arenas generated by a timed automaton. One player serves as the Environment and has an unrestricted strategy whilst the other player acts as the Controller. The Controller is restricted to only partial information about the values of the automaton's clocks and must play a strategy which may be generated by a timed automaton. Moreover, the granularity of the constraints Controller uses may be fixed in advance. These games are turn-based in that the Controller proposes a set of next moves and the Environment chooses one of them.

The problem of generating such a controller was shown to be 2-ExpTime-complete using a region construction when the granularity is fixed or the specification is deterministic [AMPS98, DM02]. It is, however, undecidable if the granularity is unspecified and either the specification is given by a nondeterministic timed automaton [DM02] or the Controller has only partial information about the state of the automaton as well as the valuation of the clocks [BDMP03].

In [CDL+07], the authors consider a variant of controller synthesis games where the Controller only has to play a strategy which is *observation based* and *stutter invariant*. These restrictions are used to ensure that any continuous strategy of the Controller can be simulated by a discrete strategy and the authors reduce the timed game to a discrete game to solve it.

The insertion channel machines we study in Chapter 5 are of independent interest as models of computation with unreliable communication. In this setting, *lossy channel machines*, for which some write operations silently fail, are more frequently studied. In

the case of the *reachability* problem, which asks whether it is possible for a system to reach a specified configuration, lossy channel machines and insertion channel machines are interreducible [OW05] and the complexity of this problem was recently characterised as non-primitive recursive [Sch02] and moreover not even multiply recursive [CS08]. The same techniques can be used to give similar complexity bounds for the termination problem for lossy channel machines, but not for insertion channel machines. It was shown in [BMO+08] that without renaming operations or fairness, the termination problem for insertion channel machines with k channels is k-ExpSpace-complete when the emptiness of those channels can be tested. The complexity of the fair termination problem for insertion channel machines therefore remains an interesting open problem in this area.

6.3 Future Work

We note that by allowing only accepting states, the class of safety alternating timed automata correspond exactly to Alpern and Schneider's semantic definition of safety properties [AS85]. In the case of the temporal logic MTL, it is clear that only safety properties are expressible under the syntactic restriction of Safety MTL, but that not all formulae of MTL which express safety properties are in Safety MTL (since every unsatisfiable formula such as $(\Diamond p \wedge \Box \neg p)$ expresses a safety property). We conjecture, however, that every safety property expressible by an MTL formula φ is expressible by an equivalent formula ψ of Safety MTL. We note that a similar result was obtained by Chang, Manna and Pnuelli, who proved that those safety properties expressible in LTL with past operators are captured exactly by a syntactic fragment [CMP92].

Our work in Chapter 5 presents a decision procedure for the fair termination problem for insertion channel machines with renaming that uses the theory of well-quasi-orderings. We conjectured that by applying recent techniques of Schnoebelen [SS11] we could bound

the complexity of our decision procedure at the level F_{ω^ω} of the extended Grzegorczyk hierarchy [LW70, CS08]. We note, however, that such a bound would be much greater than the F_4 hardness result we were able to establish, i.e., it would not even be multiply recursive.

Note that our decision procedure may be solving a harder problem than necessary since it is a *global model checking* procedure — it computes the entire set of configurations from which an insertion channel machine with renaming can begin a fair infinite run rather than simply determining whether the initial configuration is a member of that set. We thus believe that a sharper analysis based on the number of letters of the channel alphabet will yield a decision procedure whose complexity is at most F_ω. The Ackermann function is at level F_ω in this hierarchy, so the complexity of such a procedure would need to be carefully analysed to settle the open question of whether there exists a primitive recursive decision procedure for the fair termination problem.

In addition to identifying relationships between formalisms for real-time verification in order to settle questions of complexity, one should not dismiss questions of the relative expressiveness of these formalisms.

Recall that over $\langle \mathbb{N}, < \rangle$ (nondeterministic) Büchi automata and $\mathsf{MSO}(<)$ have the same expressive power [Büc62] and that in this setting, the addition of alternation to the automata adds succinctness but no increase in expressive power [MH84]. In [ORW09], the authors give a reduction from the language emptiness problem for (nondeterministic) timed automata to the satisfiability problem for $\mathsf{MSO}(<, +1)$ over $\langle \mathbb{R}_+, <, +1 \rangle$, but there can be no converse reduction since timed automata are not closed under complementation.

Alternating timed automata, which are closed under complementation, are strictly more expressive than timed automata and we were unable to directly translate the language emptiness problem to the satisfiability problem for $\mathsf{MSO}(<, +1)$ in this case. The procedure we present in Chapter 3 instead reduces the language emptiness problem for alternating

timed automata to a type of McNaughton game with an $\mathsf{MSO}(<, +1)$ winning condition. In the time-bounded case, we ultimately decide the winner of this type of game by deciding a question of $\mathsf{MSO}(<)$ satisfiability, but this is performed over the interval $[0, 1)$ and the formula to be satisfied clearly does not define the same language as the original alternating timed automaton. We therefore raise the question of whether there exists an $\mathsf{MSO}(<, +1)$ formula which defines the same language as an alternating timed automaton or whether one of these formalisms is more expressive than the other.

Finally, we turn to Church's synthesis problem as explored in Chapter 4. In this chapter we sought a real-time extension of Büchi and Landweber's theorem [BL69] which proves that for every $\mathsf{MSO}(<)$ formula $\varphi(X, Y)$, it is decidable whether there exists a causal uniformizer. Moreover, if such a uniformizer exists the algorithm computes an $\mathsf{MSO}(<)$ formula $\psi(X, Y)$ which defines it. Hence in every case where a causal uniformizer exists, it is $\mathsf{MSO}(<)$ definable. This does not hold over the reals, however, as even with the finitely-variable interpretation of $\mathsf{MSO}(<, +1)$ there exist formulae which have a causal uniformizer but where no such uniformizer can be definable in $\mathsf{MSO}(<, +1)$.

We therefore concentrate on the case of $\mathsf{MSO}(<, +1)$-definable uniformizers in the spirit of Church's original requirement that any solution to the synthesis problem be expressible as a finite-state circuit [Chu57]. The other possible generalisation is the *realizability* problem "Given an $\mathsf{MSO}(<, +1)$ formula $\varphi(X, Y)$, does there exist a causal uniformizer for φ?" and our work leaves this direction unexplored.

Appendix A

Proofs from Section 4.3

A.1 Proof of Proposition 4.9

In order to syntactically translate a formula φ which defines a signal language into $\varphi^\dagger$ which defines the corresponding word language, we will first employ a (harmless) restriction of MSO which removes all first-order quantification, then define the † translation inductively on formulae of this restricted form.

We introduce a number of auxiliary formulae to define restricted languages L_{FV} and L_ω. The most complex of these formulae is $SingE(X)$, which requires that X is a singleton and holds only at an even position, and we use it only for the word language L_ω. Since we use it only as a word formula, we use $+1$ as an abbreviation in its definition.

$$
\begin{aligned}
LT(X,Y) &\stackrel{\text{def}}{=} \forall t\,[X(t) \to \forall u(Y(u) \to t < u)] \\
Eq(X,Y) &\stackrel{\text{def}}{=} \forall t\,[X(t) \leftrightarrow Y(t)] \\
X \subseteq Y &\stackrel{\text{def}}{=} \forall t\,[X(t) \to Y(t)] \\
Sing(X) &\stackrel{\text{def}}{=} \exists t\,[X(t) \wedge \forall u(X(u) \to t = u)] \\
SingE(X) &\stackrel{\text{def}}{=} \exists E\,[E(0) \wedge \forall t\,(E(t) \to (E(t+2) \wedge \forall u(t < u < t+2 \to \neg E(u))))] \\
&\quad \wedge \exists t\,[E(t) \wedge X(t) \wedge \forall u(X(u) \to t = u)]
\end{aligned}
$$

We now define L_{FV} by the grammar

$$\varphi ::= LT(T,U) \mid Eq(T,U) \mid T \subseteq X \mid Sing(X) \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \exists X \varphi_1$$

and L_ω by the grammar

$$\varphi ::= LT(T,U) \mid Eq(T,U) \mid T \subseteq X \mid SingE(X) \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \exists X \varphi_1$$

Notice that we have removed all first-order variables. It is clear that any formula of L_{FV} or L_ω can be translated to a formula of $\mathsf{MSO}(<)$ by using the definitions above and hence the semantics of these languages do not need further definition.

We now show that restricting ourselves to formulae of L_{FV} rather than $\mathsf{MSO}(<)$ to describe finitely variable signal languages is a harmless restriction.

Lemma A.1. *For any formula $\varphi(X_1, \ldots, X_n)$ with free second-order variables amongst $X_1, \ldots, X_n$ in $\mathsf{MSO}(<)$, there exists a formula $\psi(X_1, \ldots, X_n)$ in L_{FV} such that $L_{\mathbb{R}}(\varphi) = L_{\mathbb{R}}(\psi)$.*

Proof. We obtain ψ inductively from φ. We deal with the atomic formulae as follows: replace $t < u$ with $LT(T,U)$, $t = u$ with $Eq(T,U)$ and $X(t)$ with $T \subseteq X$. Negation, conjunction and second order quantification are handled in the obvious way, leaving only the case that $\varphi = \exists t \varphi_1$. Here, we let $\psi = \exists T(Sing(T) \wedge \psi_1)$.

It is clear that this construction works as expected, as every instance where a first-order variable is used is replaced by a second-order singleton. □

We can now state a lemma which, when combined with Lemma A.1 above, is sufficient to prove Proposition 4.9.

Lemma A.2. *For any formula $\varphi(X_1, \ldots, X_n)$ with free second-order variables amongst $X_1, \ldots, X_n$ in L_{FV}, there exists a formula $\varphi^\dagger(X_1, \ldots, X_n)$ in L_ω such that $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.*

Before moving on to the proof of this lemma, we establish a technical lemma.

Lemma A.3. *For any signal f, words w, w' and sampling sequence τ such that $w = W_\tau(f)$ and $w' \sim w$, there exists τ' such that $w' = W_{\tau'}(f)$.*

Proof. Consider the signal language $\{f\}$. $\{f\}^\dagger$ is stutter-closed, so if $w = W_\tau(f)$ and $w' \sim w$, $w' \in \{f\}^\dagger$ and hence there exists τ' such that $w' = W_{\tau'}(f)$. □

Proof of Lemma A.2. We prove this lemma by induction on the structure of φ.

Base Cases

- If $\varphi(X_1, X_2) = LT(X_1, X_2)$, we let $\varphi^\dagger(X_1, X_2) \stackrel{\text{def}}{=} LT(X_1, X_2)$ also. Suppose $(w, x) \in L_{\mathbb{R}}(\varphi)^\dagger$, then there exist $(f, g) \in L_{\mathbb{R}}(\varphi)$ and τ such that $W_\tau(f, g) = (w, x)$. Since $LT(f, g)$, it is clear that $LT(w, x)$ and hence $L_{\mathbb{R}}(\varphi)^\dagger \subseteq L_{\mathbb{N}}(\varphi^\dagger)$. Suppose that $(u, v) \in L_{\mathbb{N}}(\varphi^\dagger)$, then since $LT(u, v)$ it is clear that $LT(S_{\mathbb{N}}(u), S_{\mathbb{N}}(v))$ and hence that $W_{\mathbb{N}}(S_{\mathbb{N}}(u, v))) = (u, v) \in L_{\mathbb{R}}(\varphi)^\dagger$. Hence $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.

- Similarly if $\varphi(X_1, X_2) = Eq(X_1, X_2)$ or $\varphi(X_1, X_2) = X_1 \subseteq X_2$, we take $\varphi^\dagger \stackrel{\text{def}}{=} \varphi$ and observe that $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$ by an identical argument.

- If $\varphi(X_1) = Sing(X_1)$, we let $\varphi^\dagger \stackrel{\text{def}}{=} SingE(X_1)$. Suppose $w \in L_{\mathbb{R}}(\varphi)^\dagger$, then there exists $f \in L_{\mathbb{R}}(\varphi)$ and τ such that $W_\tau(f) = w$. As $Sing(f)$ holds, $w = 0^k 1 0^\omega$ for k even. Thus $w \in L_{\mathbb{N}}(SingE(X_1))$ and $L_{\mathbb{R}}(\varphi)^\dagger \subseteq L_{\mathbb{N}}(\varphi^\dagger)$. Suppose instead that $u \in L_{\mathbb{N}}(SingE(X_1))$. Then it is clear that $u = 0^k 1 0^\omega$ for k even. Further, we observe that for g defined by

$$g(t) = \begin{cases} 0 & t < 2k \\ 1 & t = 2k \\ 0 & t > 2k \end{cases}$$

  we have $W_{\mathbb{N}}(g) = u$ and $Sing(g)$ holds, hence $u \in L_{\mathbb{R}}(\varphi)^\dagger$. Hence $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.

Inductive Cases

- If $\varphi(X_1, \ldots, X_n) = \neg\varphi_1(X_1, \ldots, X_n)$, then we let $\varphi^\dagger \stackrel{\text{def}}{=} \neg\varphi_1^\dagger$ where $\varphi_1^\dagger$ is such that $L_{\mathbb{R}}(\varphi_1)^\dagger = L_{\mathbb{N}}(\varphi_1^\dagger)$ by induction.

  Suppose $w \in L_{\mathbb{R}}(\varphi)^\dagger$, then there exists $f \in L_{\mathbb{R}}(\varphi)$ and τ such that $W_\tau(f) = w$. Note that $f \notin L_{\mathbb{R}}(\varphi_1)$ and suppose for a contradiction that $w \in L_{\mathbb{R}}(\varphi_1)^\dagger$. Then there would exist $g \in L_{\mathbb{R}}(\varphi_1)$ and τ' such that $W_{\tau'}(g) = w$. Then $f = S_\tau(w) \sim S_{\tau'}(w) = g$ and $L_{\mathbb{R}}(\varphi_1)$ is speed independent, so $f \in L_{\mathbb{R}}(\varphi_1)$, a contradiction. Hence we see $w \notin L_{\mathbb{R}}(\varphi_1)^\dagger = L_{\mathbb{N}}(\varphi_1^\dagger)$. Thus $w \in L_{\mathbb{N}}(\varphi^\dagger)$ and $L_{\mathbb{R}}(\varphi)^\dagger \subseteq L_{\mathbb{N}}(\varphi^\dagger)$.

  Suppose instead that $u \in L_{\mathbb{N}}(\varphi^\dagger)$, then $u \notin L_{\mathbb{N}}(\varphi_1^\dagger) = L_{\mathbb{R}}(\varphi_1)^\dagger$. Hence for all f and τ such that $W_\tau(f) = u$, $f \notin L_{\mathbb{R}}(\varphi_1)$. Consider $g = S_{\mathbb{N}}(u)$, then $W_{\mathbb{N}}(g) = u$ hence $g \notin L_{\mathbb{R}}(\varphi_1)$ and thus $g \in L_{\mathbb{R}}(\varphi)$. Since $W_{\mathbb{N}}(g) = u$, we have that $u \in L_{\mathbb{R}}(\varphi)^\dagger$ and $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.

- If $\varphi(X_1, \ldots, X_n) = \varphi_1(X_1, \ldots, X_n) \wedge \varphi_2(X_1, \ldots, X_n)$, then we let $\varphi^\dagger \stackrel{\text{def}}{=} \varphi_1^\dagger \wedge \varphi_2^\dagger$ where $\varphi_1^\dagger$ and $\varphi_2^\dagger$ are such that $L_{\mathbb{R}}(\varphi_1)^\dagger = L_{\mathbb{N}}(\varphi_1^\dagger)$ and $L_{\mathbb{R}}(\varphi_2)^\dagger = L_{\mathbb{N}}(\varphi_2^\dagger)$ by induction.

  Suppose $w \in L_{\mathbb{R}}(\varphi)^\dagger$, then there exists $f \in L_{\mathbb{R}}(\varphi)$ and τ such that $W_\tau(f) = w$. Note that $f \in L_{\mathbb{R}}(\varphi_1)$ and $f \in L_{\mathbb{R}}(\varphi_2)$, so $w \in L_{\mathbb{R}}(\varphi_1)^\dagger$ and $w \in L_{\mathbb{R}}(\varphi_2)^\dagger$. Hence $w \in L_{\mathbb{N}}(\varphi_1^\dagger)$ and $w \in L_{\mathbb{N}}(\varphi_2^\dagger)$ and thus $w \in L_{\mathbb{N}}(\varphi^\dagger)$ and $L_{\mathbb{R}}(\varphi)^\dagger \subseteq L_{\mathbb{N}}(\varphi^\dagger)$.

  Suppose instead that $u \in L_{\mathbb{N}}(\varphi^\dagger)$, then $u \in L_{\mathbb{N}}(\varphi_1^\dagger) = L_{\mathbb{R}}(\varphi_1)^\dagger$ hence there exist $f \in L_{\mathbb{R}}(\varphi_1)$ and τ such that $u = W_\tau(f)$ and $u \in L_{\mathbb{N}}(\varphi_2^\dagger) = L_{\mathbb{R}}(\varphi_2)^\dagger$ hence there exist $g \in L_{\mathbb{R}}(\varphi_2)$ and τ' such that $u = W_{\tau'}(g)$. Then $f = S_\tau(u) \sim S_{\tau'}(u) = g$ and $L_{\mathbb{R}}(\varphi_2)$ is speed independent, so $f \in L_{\mathbb{R}}(\varphi_2)$ hence $f \in L_{\mathbb{R}}(\varphi)$. Thus $u \in L_{\mathbb{R}}(\varphi)^\dagger$ and $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.

- If $\varphi(X_1, \ldots, X_n) = \exists X_{n+1}\varphi_1(X_1, \ldots, X_{n+1})$, we let $\varphi^\dagger$ be the formula which defines $SC(L_{\mathbb{N}}(\exists X_{n+1}\varphi_1^\dagger(X_1, \ldots, X_{n+1})))$. Let K be the signal language defined by φ_1, then by our induction hypothesis, $\varphi_1^\dagger$ defines $K^\dagger$.

Suppose $w \in L_{\mathbb{R}}(\varphi)^\dagger$, then there exists $f \in L_{\mathbb{R}}(\varphi)$ and τ such that $W_\tau(f) = w$. Then there must exist g such that $(f, g) \in K$. For any sampling sequence τ' for (f, g), we have $W_{\tau'}(f, g) \in K^\dagger = L_{\mathbb{N}}(\varphi_1^\dagger)$ hence $W_{\tau'}(f) \in L_{\mathbb{N}}(\exists X_{n+1}\varphi_1^\dagger)$. Moreover, $W_{\tau'}(f) \sim W_\tau(f) = w$, hence $w \in SC(L_{\mathbb{N}}(\exists X_{n+1}\varphi_1^\dagger))$ and $L_{\mathbb{R}}(\varphi)^\dagger \subseteq L_{\mathbb{N}}(\varphi^\dagger)$.

Suppose instead that $u \in SC(L_{\mathbb{N}}(\exists X_{n+1}\varphi_1^\dagger))$. Then there exists $v \sim u$ such that $v \in L_{\mathbb{N}}(\exists X_{n+1}\varphi_1^\dagger)$. Hence there exists x such that $(v, x) \in L_{\mathbb{N}}(\varphi_1^\dagger) = K^\dagger$. Then there exists f, g and τ such that $W_\tau(f, g) = (v, x)$ and $(f, g) \in K$. Hence $f \in L_{\mathbb{R}}(\varphi)$ and moreover there exists τ' such that $W_{\tau'}(f) = u$, so $u \in L_{\mathbb{R}}(\varphi)^\dagger$.

Hence $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.

This completes the proof of Lemma A.2. □

A.2 Proof of Proposition 4.10

In order to produce a formula $\varphi^\star$ which defines the signal language corresponding to the word language defined by φ, we consider the semantic translation from a word $w \in L_{\mathbb{N}}(\varphi)$ to a signal $f = S_\tau(w)$ for a given sequence of time points τ. We wish to relativise the first-order quantifiers present in the signal formula so that they refer only to points of τ.

We first define some auxiliary $\mathsf{MSO}(<)$ formulae. Let

$$Cont(X, t) \stackrel{\text{def}}{=} \exists u \exists v (u < t < v \wedge \forall w (u < w < v \rightarrow (X(t) \leftrightarrow X(w))))$$

and $Discont(X, t) \stackrel{\text{def}}{=} \neg Cont(X, t)$. It is clear that for any signal X and point t, $Discont(X, t)$ holds if and only if t is a discontinuity of X.

Let $\varphi(X_1, \ldots, X_n)$ be an $\mathsf{MSO}(<)$ formula with free second-order variables amongst $X_1, \ldots, X_n$ (and no free first-order variables). Assume a predicate name T that we intend to be interpreted as a sampling sequence for the signal we construct. Let $\psi(X_1, \ldots, X_n, T)$

be obtained from φ by relativising all first-order quantifiers to T as follows: replace each $\exists t\varphi_1$ with $\exists t(T(t) \wedge \varphi_1(t))$ and each $\forall t\varphi_1$ with $\forall t(T(t) \to \varphi_1(t))$.

Lemma A.4. *Let* $\mathbf{W_1}, \ldots, \mathbf{W_n}$ *be subsets of* $\mathbb{N}$ *which correspond to* ω*-words,* $\mathbf{k_1}, \ldots, \mathbf{k_m} \in \mathbb{N}$ *be natural numbers, and* $\mathbf{S_1}, \ldots, \mathbf{S_n}$ *be subsets of* $\mathbb{R}_+$ *which correspond to finitely variable signals. Let* $\mathbf{T} = \{\mathbf{t_1}, \mathbf{t_2}, \ldots\} \subseteq \mathbb{R}_+$ *be a countably infinite set such that* $\mathbf{t_i} < \mathbf{t_{i+1}}$ *for all* i *and* $\mathbf{S_i}(\mathbf{t_{k_j}}) = \mathbf{W_i}(\mathbf{k_j})$ *for* $i \leq n$, $j \leq m$. *Then*

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \textit{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \,.$$

Proof. We prove this lemma by structural induction on φ.

First we cover the cases where φ is an atomic formula.

- **Case** $\varphi(k_1, k_2) \stackrel{\text{def}}{=} k_1 < k_2$**.** In this case, $\psi(t_{k_1}, t_{k_2}, T) \stackrel{\text{def}}{=} t_{k_1} < t_{k_2}$. By the monotonic enumeration of $\mathbf{T}$, we see that $\langle \mathbb{N}, <, \mathbf{k_1}, \mathbf{k_2} \rangle \models k_1 < k_2$ if and only if $\langle \mathbb{R}_+, <, \mathbf{t_{k_1}}, \mathbf{t_{k_2}}, \mathbf{T} \rangle \models t_{k_1} < t_{k_2}$.

- **Case** $\varphi(k_1, k_2) \stackrel{\text{def}}{=} k_1 = k_2$**.** In this case, $\psi(t_{k_1}, t_{k_2}, T) \stackrel{\text{def}}{=} t_{k_1} = t_{k_2}$. We have that $\langle \mathbb{N}, <, \mathbf{k_1}, \mathbf{k_2} \rangle \models k_1 = k_2$ if and only if $\langle \mathbb{R}_+, <, \mathbf{t_{k_1}}, \mathbf{t_{k_2}}, \mathbf{T} \rangle \models t_{k_1} = t_{k_2}$.

- **Case** $\varphi(W_1, k_1) \stackrel{\text{def}}{=} W_1(k_1)$**.** In this case, $\psi(S_1, t_{k_1}, T) \stackrel{\text{def}}{=} S_1(t_{k_1})$. We have that $\langle \mathbb{N}, <, \mathbf{W_1}, \mathbf{k_1} \rangle \models W_1(k_1)$ if and only if $\langle \mathbb{R}_+, <, \mathbf{S_1}, \mathbf{t_{k_1}}, \mathbf{T} \rangle \models S_1(t_{k_1})$ since $\mathbf{S_1}(\mathbf{t_{k_1}}) = \mathbf{W_1}(\mathbf{k_1})$ by assumption.

  Next we consider the inductive cases.

- **Case** $\varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \stackrel{\text{def}}{=} \neg\varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_m)$**.**

  In this case, $\psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \stackrel{\text{def}}{=} \neg\psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T)$ where ψ_1 is obtained from φ_1 by relativising first-order quantification to T.

Then by our induction hypothesis, we may assume that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T)$$

and hence we have that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \,.$$

- **Case** $\varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \stackrel{\text{def}}{=} \left(\begin{array}{l} \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_m) \wedge \\ \varphi_2(W_1, \ldots, W_n, k_1, \ldots, k_m) \end{array} \right).$

  In this case,

$$\psi(S_1, \ldots, S_n, t_{k_1}, \ldots t_{k_m}, T) \stackrel{\text{def}}{=} \left(\begin{array}{l} \psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \wedge \\ \psi_2(S_1, \ldots, S_n, t_{k_1}, \ldots t_{k_m}, T) \end{array} \right)$$

  where ψ_1 and ψ_2 are obtained from φ_1 and φ_2 respectively by relativising first-order quantification to T.

  Then by our induction hypothesis, we may assume that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T)$$

  and

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi_2(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi_2(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \,.$$

  Hence we have that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \,.$$

- **Case** $\varphi(W_1, \ldots, W_n, k_1, \ldots k_m) \stackrel{\text{def}}{=} \exists W_{n+1} \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_m)$**.**

  In this case, $\psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \stackrel{\text{def}}{=} \exists S_{n+1} \psi_1(S_1, \ldots, S_{n+1}, t_{k_1}, \ldots, t_{k_m}, T)$ where ψ_1 is obtained from φ_1 by relativising first-order quantification to T.

  Then by our induction hypothesis, we may assume that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_{n+1}}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi_1(W_1, \ldots, W_{n+1}, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_{n+1}}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi_1(S_1, \ldots, S_{n+1}, t_{k_1}, \ldots, t_{k_m}, T)$$

  and hence we have that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \quad \text{if and only if}$$
$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \ .$$

- **Case** $\varphi(W_1, \ldots, W_n, k_1, \ldots, k_m) \stackrel{\text{def}}{=} \exists k_{m+1} \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_{m+1})$**.**

  In this case,

$$\psi(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, T) \stackrel{\text{def}}{=} \exists t_{k_{m+1}} (T(t_{k_{m+1}}) \wedge \psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_{m+1}}, T))$$

  where ψ_1 is obtained from φ_1 by relativising first-order quantification to T.

  We have that

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_m} \rangle \models \exists k_{m+1} \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_m, k_{m+1})$$
$$\text{if and only if}$$
$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_{m+1}} \rangle \models \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_{m+1})$$

for some $\mathbf{k_{m+1}}$. By our induction hypothesis,

$$\langle \mathbb{N}, <, \mathbf{W_1}, \ldots, \mathbf{W_n}, \mathbf{k_1}, \ldots, \mathbf{k_{m+1}} \rangle \models \varphi_1(W_1, \ldots, W_n, k_1, \ldots, k_{m+1}) \quad \text{holds if and only if}$$

$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_{m+1}}}, \mathbf{T} \rangle \models \psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_{m+1}}, T)$$

which holds if and only if

$$\langle \mathbb{R}_+, <, \mathbf{S_1}, \ldots, \mathbf{S_n}, \mathbf{t_{k_1}}, \ldots, \mathbf{t_{k_m}}, \mathbf{T} \rangle \models \exists t_{k_{m+1}} \left(\begin{array}{l} T(t_{k_{m+1}}) \wedge \\ \psi_1(S_1, \ldots, S_n, t_{k_1}, \ldots, t_{k_m}, t_{k_{m+1}}, T) \end{array} \right)$$

as required.

Having covered all possible cases for construction of the formula φ, we conclude that the lemma holds by induction. □

We now wish to use this rewriting to produce the finitely-variable signal language $L^\star$ when given a formula φ which defines the word language L. Let

$$\varphi^\star(X_1, \ldots, X_n) \stackrel{\text{def}}{=}$$

$$\exists T \exists EvenT \left[\begin{array}{l} EvenT(0) \wedge \psi(X_1, \ldots, X_n, T) \wedge \\ \forall t \left[T(t) \rightarrow \exists u \left(t < u \wedge T(u) \right) \right] \wedge \\ \bigwedge_{i=1}^{n} \forall t \left(\neg EvenT(t) \rightarrow Cont(X_i, t) \right) \wedge \\ \forall t \left[\begin{array}{l} EvenT(t) \rightarrow \exists u \exists v \left(\begin{array}{l} T(t) \wedge T(u) \wedge T(v) \wedge EvenT(v) \\ \wedge t < u < v \\ \wedge \forall w \left(t < w < u \rightarrow \neg T(w) \right) \\ \wedge \forall w \left(u < w < v \rightarrow \neg T(w) \right) \\ \wedge \forall w \left(t < w < v \rightarrow \neg EvenT(w) \right) \end{array} \right) \end{array} \right] \end{array} \right]$$

This formula merely requires that T be a sampling sequence for the signal encoded in variables $X_1, \ldots, X_n$ and that $\psi(X_1, \ldots, X_n, T)$ holds, where ψ is produced by the lemma above. Hence the correctness of this construction is clear and this completes the proof of Proposition 4.10.

Appendix B

Proof of Proposition 5.24

Recall that in Section 5.7.2 we wish to simulate a 1-clock safety alternating timed automaton $\mathcal{A} = (S, s_0, \Sigma, \{x\}, \delta)$ by a channel machine. We defined an insertion channel machine with renaming $\mathcal{C}_\mathcal{A}$ with set of states $\{c, d, e, f\} \times \mathbb{C}_\mathcal{A} \times \mathcal{P}(S \times Reg_\mathcal{A})$, initial state $(e, 0, \{(s_0, 0)\})$, channel alphabet $\mathcal{P}(S \times Reg_\mathcal{A}) \cup \{\triangleright\}$ and transition relation Δ (whose definition is given in Section 5.7.2).

In Proposition 5.24, we asserted the correctness of this construction as follows:

Proposition 5.24. *$L(\mathcal{A}) \neq \emptyset$ if and only if $\mathcal{C}_\mathcal{A}$ has an infinite fair run.*

In order to prove this proposition, we first formalise the notion of a configuration of $\mathcal{C}_\mathcal{A}$ representing a set of positions of an acceptance game for $\mathcal{A}$.

Given a set of acceptance game positions P, partition P into $P_0, P_1, \ldots, P_m$ as follows:

- If $(s, \nu, i), (s', \nu', i') \in P_j$, then $fract(\nu(x)) = fract(\nu'(x))$.
- If $(s, \nu, i) \in P_0$, then $fract(\nu(x)) = 0$.
- If $(s, \nu, i) \in P_j$ and $(s', \nu', i') \in P_{j'}$ for $1 \leq j < j' \leq m$, then $fract(\nu(x)) > fract(\nu'(x))$.

In other words, the partition is by equality of fractional parts of the positions' clock valuation. P_0 contains all those positions with integer valuations and the other sets are in

descending order of fractional parts.

We say that a configuration $((l, j, \alpha), \beta_1\beta_2 \dots \beta_n)$ *represents* this set of positions P if each position in P_0 is represented by an abstract position in α and there exists a monotonic injective function $f : \{1, \dots, m\} \to \{1, \dots, n\}$ such that each position in P_i is represented by an abstract position in $\beta_{f(i)}$.

We now prove the first direction of Proposition 5.24.

Proposition B.1. *If there exists $w \in L(\mathcal{A})$, then $\mathcal{C}_\mathcal{A}$ has an infinite fair run.*

Proof. Write $w = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \dots$ and observe that since $w \in L(\mathcal{A})$, Automaton has a winning strategy $\mathcal{S}$ for $G(\mathcal{A}, w)$.

We turn this winning strategy into an infinite fair run by using the σ_i for the renamings required. We must then show that we can interleave the required read and write operations correctly. We prove that we can construct a run of the required form by induction.

Suppose that we have already generated a run to the configuration $((l_i, j_i, \alpha_i), \beta_i)$ where each abstract position in $((l_i, j_i, \alpha_i), \beta_i)$ represents some position $(s, \nu, i) \in Lev_i(\mathcal{S}, w)$.

Step 1: There exists a sequence $o_1, o_2, \dots, o_{r_i}$ of operations from Op and a configuration $((l'_i, j'_i, \alpha'_i), \beta'_i)$ such that

$$((l_i, j_i, \alpha_i), \beta_i) \xrightarrow{o_1} \cdots \xrightarrow{o_{r_i}} ((l'_i, j'_i, \alpha'_i), \beta'_i)$$

and every abstract position in $((l'_i, j'_i, \alpha'_i), \beta'_i)$ represents some position $(s, \nu + \tau_{i+1} - \tau_i, i)$ where $(s, \nu, i) \in Lev_i(\mathcal{S}, w)$.

We generate this sequence of operations from the sequence of time-successor transitions required to update the clock regions of the various valuations ν represented by $((l_i, j_i, \alpha_i), \beta_i)$ to those of the valuations $\nu + \tau_{i+1} + \tau_i$ represented by $((l'_i, j'_i, \alpha'_i), \beta'_i)$.

Note that if the addition of $\tau_{i+1} - \tau_i$ would cause the region of any clock valuation represented in $((l_i, j_i, \alpha_i), \beta_i)$ to increase to (or over) an integer value, some of the o_k must be read operations by construction.

Step 2: There exists a reset set Z_{i+1} and a configuration $((l_{i+1}, j_{i+1}, \alpha_{i+1}), \beta_{i+1})$ such that

$$((l'_i, j'_i, \alpha'_i), \beta'_i) \xrightarrow{R_{\sigma_{i+1}}^{Z_{i+1}}} ((l_{i+1}, j_{i+1}, \alpha_{i+1}), \beta_{i+1})$$

and every abstract position in this configuration represents some $(s, \nu, i+1) \in Lev_{i+1}(\mathcal{S}, w)$.

We choose Z_{i+1} and the corresponding successor positions by following $\mathcal{S}$ since $\mathcal{S}$ is a winning strategy for $G(\mathcal{A}, w)$ and hence there exist successor positions for each position in $Lev_i(\mathcal{S}, w)$.

The combination of these steps completes the inductive part of the proof and ensure that we can generate an infinite run of $\mathcal{C}_\mathcal{A}$. We observe that this run is fair because at least one read operation occurs infinitely often as part of the operations in Claim 1, hence $\triangleright$ must be read (and subsequently written) infinitely many times thus there are infinitely many cycles. □

Before we prove the other direction of Proposition 5.24, we prove a lemma which allows us to extract a time delay from each non-renaming transition of $\mathcal{C}_\mathcal{A}$.

Lemma B.2. *For each set of positions P and transition $((l, j, \alpha), \beta) \xrightarrow{o} ((l', j', \alpha'), \beta')$ of $\mathcal{C}_\mathcal{A}$ such that $((l, j, \alpha), \beta)$ represents P there exists $t \in \mathbb{R}_+$ such that $((l', j', \alpha'), \beta')$ represents the set of positions $\{(s, \nu + t, i) \mid (s, \nu, i) \in P\}$. Moreover, if $l = d$ and $l' = c$, $t > 0$.*

Proof. We prove this lemma by analysing the possible transitions of $\mathcal{C}_\mathcal{A}$.

If $o = \triangleright?$ or $o = \triangleright!$, the applicable transitions are

$$((c, j, \alpha), \triangleright \cdot \beta) \xrightarrow{\triangleright?} ((e, j, \alpha), \beta), \qquad ((d, j, \alpha), \triangleright \cdot \beta) \xrightarrow{\triangleright?} ((f, j, \alpha), \beta)$$
$$((c, j, \alpha), \beta) \xrightarrow{\triangleright?} ((e, j, \alpha), \beta), \qquad ((d, j, \alpha), \beta) \xrightarrow{\triangleright?} ((f, j, \alpha), \beta)$$
$$((e, j, \alpha), \beta) \xrightarrow{\triangleright!} ((c, j+1, \alpha), \beta \cdot \triangleright) \quad \text{and} \quad ((f, j, \alpha), \beta) \xrightarrow{\triangleright!} ((d, j+1, \alpha), \beta \cdot \triangleright).$$

Since none of these transitions affect α or the channel letters which contain abstract positions, we simply choose $t = 0$ and leave P unchanged also.

The only case in which positions represented by a channel letter are affected is when $o = \alpha?$, the transition is of the form $((l, j, \emptyset), \alpha \cdot \beta) \xrightarrow{\alpha?} ((c, j, Succ(\alpha)), \beta)$ (i.e. it is not a read error) and the positions in P_1 are represented by abstract positions in the channel letter α. In this case we choose $(s, \nu, i) \in P_1$ and take $t = 1 - fract(\nu(x))$. Then it is clear that every position in $\{(s, \nu + t, i) \mid (s, \nu, i) \in P_1\}$ is represented by an abstract position in $Succ(\alpha)$ and has $fract(\nu + t(x)) = 0$. Moreover, for any position $(s', \nu', i') \in P_p$ with $p > 1$, $fract(\nu'(x)) < fract(\nu(x))$ so $\lfloor \nu' \rfloor = \lfloor \nu' + t \rfloor$. Then if (s', ν', i') is represented by an abstract position $(s, n^+) \in \beta_k$, $(s', \nu' + t, i')$ is also represented by (s, n^+). β therefore still represents the set of positions $P_2 \dots P_m$.

The other possible cases for transitions are

$$((d, j, \emptyset), \beta) \xrightarrow{nop} ((c, j, \emptyset), \beta), \qquad ((l, j, \alpha), \beta) \xrightarrow{Succ(\alpha)!} ((c, j, \emptyset), \beta \cdot Succ(\alpha))$$

$$((l, j, \emptyset), \beta) \xrightarrow{\alpha?} ((c, j, Succ(\alpha)), \beta), \quad \text{or} \quad ((l, j, \emptyset), \alpha \cdot \beta) \xrightarrow{\alpha?} ((c, j, Succ(\alpha)), \beta)$$

when the positions in P_1 are represented by abstract positions in a channel letter β_k for some k. In each of these cases, we choose $t > 0$ such that none of the positions in $P_1, \dots, P_m$ would have their clock region increased. Since all positions in $P_1, \dots, P_m$ are represented by the channel contents β none of their clock valuations are integers. It is therefore possible to choose such a t by the density of time. Note that for a position $(s, \nu, i) \in P_0$ with $\nu(x) = n$, (s, ν, i) is represented by α if and only if $(s, \nu + t, i)$ is represented by $Succ(\alpha)$. Moreover $fract(\nu + t(x)) < fract(\nu' + t(x))$ for any position $(s', \nu', i') \in P_k$ with $k > 0$ since $\lfloor \nu' \rfloor = \lfloor \nu' + t \rfloor$ as the clock region of $(s', \nu' + t, i')$ is the same as the region of (s', ν', i'). Hence this t correctly updates the representation of positions in P_0 and it is simple to see that the representation of all other positions is likewise correctly updated. □

We can now prove the second direction of Proposition 5.24.

Proposition B.3. *If $\mathcal{C}_\mathcal{A}$ has an infinite fair run, then there exists $w \in L(\mathcal{A})$.*

Proof. By construction, every infinite fair run of $\mathcal{C}_\mathcal{A}$ has infinitely many renaming operations (and infinitely many read operations). Say that the configuration immediately before the

ith such renaming is $((c, k_{i-1}, \alpha'_{i-1}), \beta'_{i-1})$ and the configuration immediately following it is $((d, 0, \alpha_i), \beta_i)$. Every infinite fair run therefore has the form

$$\begin{aligned}
&((e,0,\alpha_0),\beta_0) \xrightarrow{o^0_1} \cdots \xrightarrow{o^0_{n_0}} ((c,k_0,\alpha'_0),\beta'_0) \xrightarrow{R^{Z_1}_{\sigma_1}} ((d,0,\alpha_1),\beta_1) \xrightarrow{o^1_1} \cdots \\
&\qquad\qquad \vdots \\
&((d,0,\alpha_i),\beta_i) \xrightarrow{o^i_1} \cdots \xrightarrow{o^i_{n_i}} ((c,k_i,\alpha'_i),\beta'_i) \xrightarrow{R^{Z_{i+1}}_{\sigma_{i+1}}} ((d,0,\alpha_{i+1}),\beta_{i+1}) \xrightarrow{o^{i+1}_1} \cdots \\
&\qquad\qquad \vdots
\end{aligned}$$

for some $o_j \in Op$ (i.e. not a renaming). We use the sequence of letters $\sigma = \sigma_1\sigma_2\ldots$ from the renaming operations as the sequence of letters in w and we must infer the corresponding timestamps τ_i from the sequences of read and write operations which intersperse the renamings. We generate these τ_i in combination with a strategy $\mathcal{S}_i$ for Automaton in the first i rounds by induction on the round number.

Suppose we have generated the first i timed events of w as $(\sigma_1, \tau_1)\ldots(\sigma_i, \tau_i)$ and the corresponding i levels of Automaton's strategy $\mathcal{S}_i$ such that every position $(s, \nu, i) \in Lev_i(\mathcal{S}_i, w)$ is represented by $((l_i, j_i, \alpha_i), \beta_i)$. Recall from the above that $Lev_i(\mathcal{S}_i, w)$ is well defined.

Step 1: There exists a delay δ_{i+1} such that for every position $(s, \nu, i) \in Lev_i(\mathcal{S}, w)$, the position $(s, \nu + \delta_{i+1}, i)$ is represented by $((l'_i, j'_i, \alpha'_i), \beta'_i)$.

This δ_{i+1} is computed by summing the t provided by Lemma B.2.

Step 2: Taking $w_{i+1} = (\sigma_{i+1}, \tau_i + \delta_{i+1})$, we can extend $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$ such that every $(s, \nu, i+1) \in Lev_{i+1}(\mathcal{S}_{i+1}, w)$ is represented by $((l_{i+1}, j_{i+1}, \alpha_{i+1}), \beta_{i+1})$.

We choose such a $\mathcal{S}_{i+1}$ by setting the model chosen for each position in $Lev_i(\mathcal{S}_i, w)$ to the set of successor positions chosen for it by the transition with operand $R^{Z_{i+1}}_{\sigma_{i+1}}$ in our infinite fair run of $\mathcal{C}_\mathcal{A}$.

These two steps complete the inductive part of the proof. We note that the by following the generated strategy $\mathcal{S}$, Automaton can ensure that every play of $G(\mathcal{A}, w)$ is infinite and hence winning for him (since $\mathcal{A}$ is a safety alternating timed automaton). We must finally

show that the sequence of timestamps $\tau = \tau_1\tau_2\ldots$ is non-Zeno (i.e. unbounded). We see this because every read of a channel letter which represents (s, ν, i) advances $\nu(x)$ to the next integer value; since the run of $\mathcal{C}_\mathcal{A}$ is fair, such reads occur infinitely often there are infinitely many (non-overlapping) sequences $\tau_k, \tau_{k+1}, \ldots, \tau_{k'}$ of consecutive timestamps where $\tau_{k'} - \tau_k = 1$. □

This completes the proof of Proposition 5.24.

Bibliography

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ADOW05] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proceedings of the IFAC Symposium on System Structure and Control*, pages 469–474, 1998.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[BCM05] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2003.

[BHKH05] Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Boudewijn R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2–26, 2005.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[BMO+08] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James Worrell. On termination for faulty channel machines. In *STACS*, volume 1 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The cost of punctuality. In *LICS*, pages 109–120. IEEE Computer Society, 2007.

[Büc60] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift Math. Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[Büc62] J. Richard Büchi. On a decision method in restricted second-order arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CDL+07] Franck Cassez, Alexandre David, Kim Guldstrand Larsen, Didier Lime, and Jean-François Raskin. Timed control with observation based and stuttering invariant strategies. In *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2007.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[Chu57] Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50, Cornell University, Ithaca, N.Y., 1957.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[CMP92] Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *ICALP* [DBL92], pages 474–486.

[CS08] Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *LICS*, pages 205–216. IEEE Computer Society, 2008.

[dAFH+03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.

[DBL92] *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *Lecture Notes in Computer Science*. Springer, 1992.

[DG08] Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

[DGRR09] Laurent Doyen, Gilles Geeraerts, Jean-François Raskin, and Julien Reichert. Realizability of real-time logics. In *FORMATS*, volume 5813 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2009.

[DM02] Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582. Springer, 2002.

[Far02] Berndt Farwer. ω-automata. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*, pages 265–274. Springer Berlin / Heidelberg, 2002.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, pages 60–65. ACM, 1982.

[Grz53] Andrzej Grzcgorczyk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4, 1953.

[Her98] Philippe Herrmann. Timed automata and recognizability. *Information Processing Letters*, 65:313–318, 1998.

[Hig52] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(2):326–336, 1952.

[HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In Frank Thomson Leighton and Allan Borodin, editors, *STOC*, pages 373–382. ACM, 1995.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *ICALP* [DBL92], pages 545–558.

[HPS83] David Harel, Amir Pnueli, and Jonathan Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222 – 243, 1983.

[HR04] Yoram Hirshfeld and Alexander Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1), 2004.

[HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[HST09] Paul Hänsch, Michaela Slaats, and Wolfgang Thomas. Parametrized regular infinite games and higher-order pushdown strategies. In *Proceedings of FCT 09*, volume 5699 of *Lecture Notes in Computer Science*. Springer, 2009.

[JORW10] Mark Jenkins, Joël Ouaknine, Alexander Rabinovich, and James Worrell. Alternating timed automata over bounded time. In *LICS*, pages 60–69. IEEE Computer Society, 2010.

[JORW11] Mark Jenkins, Joël Ouaknine, Alexander Rabinovich, and James Worrell. The Church synthesis problem with metric. In *CSL*, volume 12 of *LIPIcs*, pages 307–321. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KZ06] Joost-Pieter Katoen and Ivan S. Zapreev. Safe on-the-fly steady-state detection for time-bounded reachability. In *QEST*, pages 301–310. IEEE Computer Society, 2006.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[LS98] Shmuel Lifsches and Saharon Shelah. Uniformization and Skolem functions in the class of trees. *The Journal of Symbolic Logic*, 63(1):103–127, 1998.

[LW70] M. H. Löb and S. S. Wainer. Hierarchies of number-theoretic functions. i. *Archive for Mathematical Logic*, 13:39–51, 1970.

[LW05] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. In *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[McN65] Robert McNaughton. Finite-state infinite games. Project MAC report, MIT, Cambridge, Mass., September 1965.

[McN93] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34(5):1045–1079, 1955.

[MH84] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Mil00] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

[Min61] Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *The Annals of Mathematics*, 74(3):pp. 437–455, 1961.

[ORW09] Joël Ouaknine, Alexander Rabinovich, and James Worrell. Time-bounded verification. In *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 496–510. Springer, 2009.

[OW04] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS*, pages 54–63. IEEE Computer Society, 2004.

[OW05] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197. IEEE Computer Society, 2005.

[OW06a] Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[OW06b] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[PW09] Pawel Parys and Igor Walukiewicz. Weak alternating timed automata. In *ICALP (2)*, volume 5556 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2009.

[Rab02] Alexander Rabinovich. Finite variability interpretation of monadic logic of order. *Theoretical Computer Science*, 275(1-2):111–125, 2002.

[Rab07a] Alexander Rabinovich. The Church synthesis problem with parameters. *Logical Methods in Computer Science*, 3(4), 2007.

[Rab07b] Alexander Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. *Information and Computation*, 205(6):870–889, 2007.

[RR94] Olivier Roux and Vlad Rusu. Verifying time-bounded properties for electre reactive programs with stopwatch automata. In *Hybrid Systems*, volume 999 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 1994.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sch02] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[She75] Saharon Shelah. The monadic theory of order. *The Annals of Mathematics*, 102(3):379–419, 1975.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *Proceedings of STOC 73*, pages 1–9. ACM, 1973.

[SS11] Sylvain Schmitz and Philippe Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 2011.

[Tho08] Wolfgang Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008.

[Tra95] Boris A. Trakhtenbrot. Origins and metamorphoses of the trinity: Logic, nets, automata. In *LICS*, pages 506–507. IEEE Computer Society, 1995.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.