Temporal Logics Beyond Regularity

Martin Lange

BRICS Report Series RS-07-666

ISSN 0909-0878 July 2007

Temporal Logics Beyond Regularity

Martin Lange

July 2007

Abstract

Non-regular program correctness properties play an important role in the specification of unbounded buffers, recursive procedures, etc. This thesis surveys results about the relative expressive power and complexity of temporal logics which are capable of defining non-regular properties. In particular, it features Propositional Dynamic Logic of Context-Free Programs, Fixpoint Logic with Chop, the Modal Iteration Calculus, and Higher-Order Fixpoint Logic.

Regarding expressive power we consider two classes of structures: arbitrary transition systems as well as finite words as a subclass of the former. The latter is meant to give an intuitive account of the logics' expressive powers by relating them to known language classes defined in terms of grammars or Turing Machines.

Regarding the computational complexity of temporal logics beyond regularity we focus on their model checking problems since their satisfiability problems are all highly undecidable. Their model checking complexities range between polynomial time and non-elementary.

Habilitationsschrift

Temporal Logics Beyond Regularity

Martin Lange

LFE Theoretische Informatik
Department "Institut für Informatik"
Ludwig-Maximilians-Universität München

and

Department of Computer Science
Aarhus University

July 11, 2007

Synopsis

This thesis provides an overview of results regarding the expressive power and complexities of temporal logics for non-regular properties. In particular, it summarises the main results of the following papers, as well as related results by other authors.

1. M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

2. M. Lange. The alternation hierarchy in fixpoint logic with chop is strict too. *Information and Computation*, 204(9):1346–1367, 2006.

3. M. Lange. Three notes on the complexity of model checking fixpoint logic with chop. To appear in *R.A.I.R.O. – Theoretical Informatics and Applications*, 2006.

4. M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Information Processing Letters*, 100(2):72–75, 2006.

5. M. Lange and R. Somla. The complexity of model checking higher order fixpoint logic. In *Proc. 30th Int. Symp. on Math. Foundations of Computer Science, MFCS'05*, volume 3618 of *Lect. Notes in Computer Science*, pages 640–651, Gdansk, Poland, 2005. Springer-Verlag.

6. M. Lange. Alternating context-free languages and linear time μ-calculus with sequential composition. In *Proc. 9th Workshop on Expressiveness in Concurrency, EXPRESS'02*, volume 68.2 of *El. Notes in Theor. Computer Science*, pages 71–87, 2002. Elsevier Science.

7. R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3(2:7):1–33, 2007.

Acknowledgments

First of all I wish to thank my wife Becky and my daughters Annika and Sophie for giving me the support that allows me to work as a scientist and, consequently, to work on this topic for quite some time. I am particularly grateful to them for showing understanding and sympathy and putting their own needs aside, for example when it came to doing two major moves in the last years – from Edinburgh to Munich and then on to Århus. I should mention though that Annika wasn't really given much of a choice there, and Sophie even less so because we simply assumed that unborn babies don't mind moving house that much.

I also want to thank the numerous colleagues who have contributed to this work by discussing various issues covered in this thesis with me: Stephan Kreutzer, Rafał Somla, Roland Axelsson, Damian Niwiński, Klaus Aehlig, Paul Hunter, Joachim Baran, Jan Johannsen, Hans-Wolfgang Loidl, Martin Hofmann, Moshe Vardi, Marcin Jurdziński, Markus Müller-Olm, Colin Stirling, Igor Walukiewicz, Alexander Okhotin, Martin Otto, anyone I may have forgotten, and the anonymous referees of the publications listed above.

I want to thank Martin Hofmann, Fred Kröger, and Klaus Schulz for agreeing to act as my mentors as well as for the input and suggestions that they have provided me with.

Finally, I want to thank Rajeev Alur, Damian Niwiński, Martin Hofmann and Fred Kröger again for reviewing this thesis.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise, and that this work has not been submitted for any other degree or professional qualification.

My own contribution to the co-authored papers listed above can roughly be specified as 70% of (4), 30% of (5), and 80% of (7).

The translation in the proof of Thm. 7.3 in Appendix A.1 is a result of a joint effort between Klaus Aehlig and myself. Roland Axelsson deserves merits for working out the details of the observation that conjunctive grammars and alternating context-free grammars are actually the same (Section 3.2). Example 5.4 in Section 5.1 is due to Paul Hunter. Examples 5.5 in the same section and 6.3 in Section 6.2 have been found together with Stephan Kreutzer.

Contents

1 Introduction **1**
- 1.1 Temporal Logics in Computer Science 1
- 1.2 Model Checking 2
- 1.3 Non-Regular Program Correctness Properties 3
- 1.4 Outline 5

2 Preliminaries **8**
- 2.1 Labeled Transition Systems and Finite Words 8
- 2.2 Properties of Labeled Transition Systems 11
- 2.3 The Chomsky Hierarchy 12
- 2.4 The Modal μ-Calculus 12
- 2.5 Descriptive Complexity 14
- 2.6 Satisfiability and Validity 15
- 2.7 Finitary Axiomatisations and the Finite Model Property 16

3 The Space Above Context-Free Languages **20**
- 3.1 Closures of Language Classes 20
- 3.2 Alternating Context-Free Languages 21
- 3.3 A General Inexpressivity Theorem 26

4 Propositional Dynamic Logic of Context-Free Programs **29**
- 4.1 Syntax and Semantics 29
- 4.2 Expressive Power 30
- 4.3 Complexity Results 31

5 Fixpoint Logic with Chop **33**
- 5.1 Syntax and Semantics 33
- 5.2 Expressive Power 37
- 5.3 Complexity Results 39

6 The Modal Iteration Calculus **41**
6.1 Inflationary Fixpoint Iterations 41
6.2 Syntax and Semantics 42
6.3 Complexity Results 44
6.4 Expressive Power 45

7 Higher-Order Fixpoint Logic **47**
7.1 Syntax and Semantics 47
7.2 Expressive Power and Complexity Results 51

8 Conclusions **54**
8.1 Summary 54
8.2 The Family of Temporal Logics Beyond Regularity . . . 55
8.3 The Complexity of Model Checking a Non-Regular Property 57
8.4 Open Questions 57
8.5 Further Work 61

A Unpublished Proofs **74**
A.1 From the Modal Iteration Calculus to Higher-Order Fixpoint Logic 74

Chapter 1

Introduction

1.1 Temporal Logics in Computer Science

Temporal logics have their origin in philosophy as variants of modal logics which enable reasoning about necessities and possibilities. Prior developed *Tense Logic* – the root of what is nowadays called *Temporal Logic* – building on apparent logical connections between necessities and time [Pri57].

Kripke suggested to use node- and edge-labeled directed graphs – now known as *Kripke structures* – to give modal logics a sound semantics [Kri63]. In computer science, such structures also occur in the operational semantics of state-based programs.[1] Hence, modalities can be used to reason about program behaviour.

This has been observed in the seventies when temporal logics – as modal logics with some reference to the evolving of time – have been introduced into computer science. One of the first examples was *Propositional Dynamic Logic* (PDL) by Fisher and Ladner who equipped multimodal logic with a Kleene algebra of accessibility relations in order to reason about programs that are formed using non-deterministic choice, composition and iteration [FL79].

Roughly at the same time Pnueli suggested to use *Linear Time Temporal Logic* (LTL) for program verification purposes. Kamp's result about the equi-expressiveness between LTL and first-order logic on certain infinite linear orders [Kam68] made LTL a default candidate. However, LTL cannot reason about internal non-deterministic choices of programs. This led to the introduction of *branching time temporal logics* simi-

[1]We will use the term *program* to denote both hard- and software.

lar to PDL, by Ben-Ari/Manna/Pnueli [BAPM83], and Clarke/Emerson [EC82]. Later, Emerson and Halpern tried to unify the linear time and the branching time approach in the introduction of CTL* which subsumes both LTL and CTL [EH86].

At the beginning of the eighties, Kozen developed the *modal μ-calculus* ($\mathcal{L}_\mu$) which simply extends multi-modal logic with extremal fixpoint quantifiers [Koz83]. It subsumes all of the logics mentioned above with uniform translations for PDL and CTL. The embedding of CTL* and with it LTL into $\mathcal{L}_\mu$ is exponential and slightly more involved [Dam94].

The main research focus was on the satisfiability problem for these logics. Not only can several problems about program specifications, for instance subsumption, be reduced to satisfiability. It is also the case that the satisfiability problem for these logics have reasonably low complexities compared to predicate logics for instance: PDL and CTL are EXPTIME-complete [FL79, Pra78, EH85], LTL is PSPACE-complete [SC85], CTL* is 2-EXPTIME-complete [EJ00, VS85], and the modal μ-calculus is also EXPTIME-complete [EJ00].

1.2 Model Checking

In the early '80s another verification method was introduced by Clarke, Emerson, Sistla and Queille, Sifakis: *model checking* [CES83, QS82]. Here, the program to be verified is abstracted into a Kripke structure and the property to be checked is formalised in a temporal logic. Model checking is then simply the logical problem of determining whether or not a given interpretation satisfies a given formula. The computational complexity of the model checking problem is usually below that of the satisfiability problem for a given logic: for PDL and CTL it is P-complete [FL79, EH85], for LTL it remains PSPACE-complete (when using the implicit "for all paths" quantification) [SC85], for CTL* it is also PSPACE-complete [EL87], and for the modal μ-calculus it is P-hard and in UP$\cap$co-UP [Jur98].

The main model checking techniques that are used these days are automata-theoretic [Var96, KVW00], tableau- or game-based [SS98, Cle90], graph-colouring methods [And94, CS92], and, lately, symbolic methods [McM93, CBRZ01].

By now, the phrases *temporal logics* and *model checking* have become inseparably linked. Model checking temporal logics has become the main method for automatic program verification [CGP99], and verification tools that implement model checking algorithms have revealed numerous errors in models of soft- and hardware.

The main threat to model checking as a successful verification method is the *state-space explosion problem*. The global state-space of a system that is composed of several components is exponential in the number of components. Hence, model checking becomes difficult because of the sheer size of the input that a model checker usually has to face, and theoretical tractability results become useless. A lot of research effort has been put into tackling this problem ranging from various optimisations to the development of new techniques like *abstraction* [CC77, DGG97] or *bounded model checking* [CBRZ01] which sacrifice completeness of the model checking results in favour of smaller inputs.

1.3 Non-Regular Program Correctness Properties

Another drawback of model checking as it has mainly been developed and used so far is expressive power. The modal μ-calculus is equi-expressive to the bisimulation-invariant fragment of Monadic Second Order Logic over trees or arbitrary Kripke structures [JW96]. Therefore, every property that is expressible in the modal μ-calculus or its fragments mentioned above can also be checked by a finite Rabin tree automaton. Thus, it is only a *regular property*. There are, however, many interesting correctness properties of programs that are not regular. Examples include *uniform inevitability* [Eme87] which states that a certain event occurs globally at the same time in all possible runs of the system; counting properties like "at any point in a run of a protocol there have never been more *send*- than *receive*-actions"; formulas saying that an unbounded number of data does not lose its order during a transmission process; or properties making structural assertions about their models like being bisimilar to a linear time model, etc.

Here we aim at forming the theoretical foundations for the verification of non-regular properties. We introduce several logics that are capable of expressing such properties, compare them w.r.t. their expressive powers and the complexities of their model checking problems on finite structures. Their satisfiability problems will receive less attention due to undecidability. The same applies to the model checking problem on infinite structures. Undecidability of the satisfiability problem is also the cause for the lack of the finite model property for these logics.

It is tempting to say that model checking finite structures only with formulas that do not have the finite model property is meaningless. This is not the case though. On a given finite structure logics for regular properties suffice to describe every possible property. But this is only the case because a given structure either has the property at hand or not. Hence, with this argumentation one would consequently have to propose the boolean logic of two values $\mathtt{tt}$ and $\mathtt{ff}$ as the ultimatively sufficient logic for model checking.

Given a *class* of finite structures it should be clear that regular specification formalisms do not always suffice to describe any property, for otherwise the Chomsky hierarchy would collapse to that level. Hence, model checking non-regular properties of finite transition systems is not meaningless.

On the other hand, if there is an a priori bound on the sizes of each considered structure then regular properties do suffice. This is reflected in the fact that every language $L \subseteq \Sigma^*$ with a maximal word length below some fixed n is regular. But even if regular specification languages suffice for some reason, non-regular ones can provide a distinct advantage over regular ones.

Example 1.1 For two words u und w over some alphabet Σ we write $u \preceq w$ to indicate that u is a prefix of w. We also use $|w|_a$ for some $a \in \Sigma$ to denote the number of occurrences of the letter a in w.

Let $\Sigma = \{\mathtt{in}, \mathtt{out}\}$ and consider the language

$$L \;=\; \{w \mid \forall u \in \Sigma^*, u \preceq w \Rightarrow 0 \leq |u|_{\mathtt{in}} - |u|_{\mathtt{out}}\}$$

It is not hard to see that L is not a regular language. It is (deterministic) context-free though. Hence, so is its complement $\overline{L}$ which is, for example, generated by the following grammar G.

$$\begin{array}{rcl} S & \to & A\,\mathtt{out}\,U \\ A & \to & \epsilon \mid \mathtt{in}\,T\,A \\ T & \to & \mathtt{out} \mid \mathtt{in}\,T\,T \\ U & \to & \epsilon \mid \mathtt{in}\,U \mid \mathtt{out}\,U \end{array}$$

Suppose $\mathtt{in}$ and $\mathtt{out}$ are interpreted as respective actions of a buffer. Then $\overline{L}$ describes those runs of a buffer that cause an underflow, and L could be used in linear time model checking to verify the absence of such underflows and, thus, the partial correctness of a buffer implementation.

An unlimited buffer is an infinite state-space system. Hence, it probably cannot be verified against L automatically. Consider therefore bounded buffers that can contain at most n elements. They clearly give rise to finite state-space systems, and the non-underflowing runs of a buffer of size n are described by the languages

$$L_n \;=\; \{w \mid \forall u \in \Sigma^*, u \preceq w \Rightarrow 0 \leq |u|_{\mathtt{in}} - |u|_{\mathtt{out}} \leq n\}$$

These languages are regular for every $n \in \mathbb{N}$. It is not hard to see that they are recognised by deterministic finite automata with $n + 2$ states.

Note that $L_n \subseteq L$ for all $n \in \mathbb{N}$. Hence, any buffer implementation that is correct w.r.t. L is also correct w.r.t. the corresponding L_n. This means that any underflow that is detected using a regular specification language for some L_n will also be detected using the non-regular specification language $\overline{L}$. The latter provides two distinct advantages, though.

- Using regular specification formalisms for the detection of buffer underflows requires knowledge about the exact buffer capacity of the underlying system and is therefore not suitable for black-box verification. Using the non-regular property $\overline{L}$, however, eliminates the need for this kind of knowledge.

- The grammar G above for $\overline{L}$ is fixed and does not depend on the underlying system. The sizes of the deterministic finite automata for the languages $\overline{L_n}$ grow linearly in n. In some cases, model checking with a fixed formula may be easier than model checking when the formula is part of the input as well. Fixed formulas, for example, can be provided in libraries, and there may be advantages regarding complexity and optimisations as well.

1.4 Outline

Chp. 2 starts with the basic definitions of labeled transition systems, a synonym for Kripke structures and the common class of interpretations for all temporal logics. We briefly recall the modal μ-calculus for two reasons. The first is its exposed status among temporal logics for regular properties, subsuming basically all of them. This makes it a good yardstick which other logics (of non-regular properties) can be measured against in terms of their expressive power. Secondly, three of these logics considered explicitly in later chapters are in fact simple extensions of the modal μ-calculus.

To provide further information about the expressive power of a temporal logic we consider a restricted class of models: finite words, resp. their usual encoding as linear and finite labeled transition systems. This allows us to embed new formalisms into the well-known Chomsky hierarchy for which there is no equivalent framework in terms of arbitrary transition systems.

Chp. 2 furthermore gives a very brief introduction to the main principle of descriptive complexity theory: the *data complexity* of a logic. This allows us to compare logics to known complexity classes w.r.t. to expressiveness. The final sections of this chapter explain in more detail why we hardly consider a logic's satisfiability problem here, and why we take the model-theoretic approach to temporal logics rather than the proof-theoretic one which characterises logics through sound and complete axiomatisations and their properties.

Chp. 3 is entirely devoted to logics of finite words where the term "logic" is meant in the broadest possible way as a formal system whose elements can be interpreted over words. As mentioned above, this is true for complexity classes with Turing Machines that recognised a word, but also for formal grammars which generate these words. Generation, recognition, and satisfaction of course are only different ways of looking at the same concept.

Chp. 4 introduces the oldest temporal proper logic of non-regular properties: *Propositional Dynamic Logic of Context-Free Programs* by Harel, Pnueli and Stavi [HPS83] – originally called *PDL of Non-Regular Programs*. It extends ordinary PDL by employing an infinite set of accessibility relations that is formed using context-free grammars rather than a Kleene algebra. It has not received much attention since its introduction which is mainly due to the undecidability of its satisfiability problem and the focus on satisfiability in early years.

Chp. 5 is about a much younger temporal logic of non-regular properties: *Fixpoint Logic with Chop* (FLC) by Müller-Olm. It extends the modal μ-calculus with a sequential composition operator. Note that formulas of modal logic (with or without fixpoints) look like right-linear grammars with modal operators as terminal symbols. It is therefore not surprising that μ-calculus-expressible properties are only regular. With sequential composition it becomes possible to put modal operators behind variables and, hence, to achieve non-regular effects comparable to context-free grammars.

Chp. 6 contains a different extension of the modal μ-calculus: the

Modal Iteration Calculus (MIC) by Dawar, Grädel and Kreutzer. They simply replace the least and greatest fixpoint operators over monotone functionals in the μ-calculus by inflationary and deflationary fixpoints over arbitrary functionals. This is inspired by work on fixpoint extensions of first-order logic. However, the results are different. While first-order logic with inflationary fixpoints is only as expressive as first-order logic with least fixpoints [GS86, Kre04], MIC properly extends the expressive power of the modal μ-calculus.

Chp. 7 introduces the most expressive of the logics considered here: *Higher-Order Fixpoint Logic* (HFL) by Viswanathan and Viswanathan [VV04]. It consequently follows the idea used to explain the semantics of FLC: lifting predicates as needed for the modal μ-calculus to *predicate transformers* for which sequential composition is easily defined. HFL's high expressive power is then obtained by employing higher-order predicate transformers, i.e. functions that take predicate transformers as arguments, etc. Syntactically this simply means that HFL is a hybrid of the modal μ-calculus and the *simply typed λ-calculus* with one ground type and the function arrow as the only type constructor.

This thesis is meant to provide an overview of the theoretical results on temporal logics for non-regular properties w.r.t. expressive power and complexity. It only presents the main theorems on the respective logics and gives some short explanations about the proof technique used to obtain this result, its difficulty etc. Detailed proofs are omitted for space considerations but also in order not to spoil the overview character, and can be found in the accompanying cited papers. Appendix A contains a detailed unpublished proof concerning the relationship between MIC and HFL.

Chp. 8 concludes by summarising the results of the previous chapters in tabular and graphical form. It also lists questions about these logics that remain open to date, and suggests further research with respect to the applicability of temporal logics beyond regularity in automatic program verification.

Chapter 2

Preliminaries

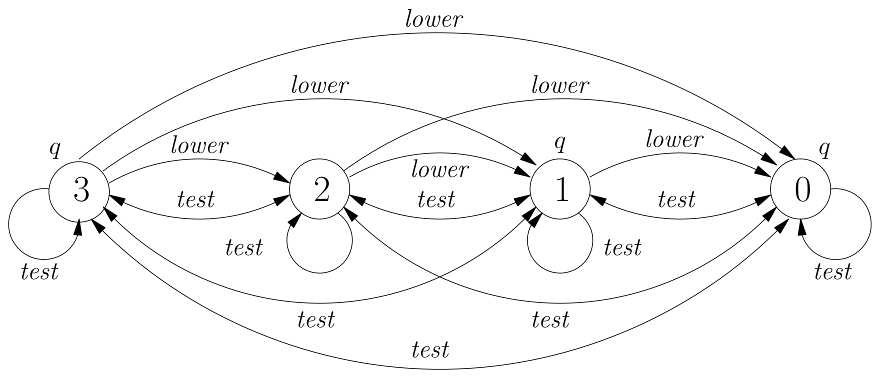
2.1 Labeled Transition Systems and Finite Words

All the logics considered here are interpreted over directed, node- and edge-labeled graphs – called *Kripke structures* or *labeled transition systems*.

Definition 2.1 Let Σ be a finite alphabet, $\mathcal{P}$ be an at most countably infinite set of atomic propositions. A *labeled transition system* (LTS) is a tuple $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ where

- $\mathcal{S}$ is a set (called states),
- $\xrightarrow{a} \subseteq \mathcal{S} \times \mathcal{S}$ for every $a \in \Sigma$ is a binary relation on states, and
- $L : \mathcal{S} \to 2^{\mathcal{P}}$ is a function that assigns to each state the set of propositions that hold true in it.

Example 2.1 The graph below depicts a simple LTS with $\mathcal{S} = \{0, 1, 2, 3\}$, and two transition relations $\xrightarrow{lower}$ and $\xrightarrow{test}$ over the singleton set $\mathcal{P} = \{q\}$ of atomic propositions.



This is taken from [ALS07] where such transition systems are created in order to establish a complexity-theoretic hardness result for the model checking problem of HFL as introduced in Chp. 7.

Finite words over an alphabet Σ are usually defined as finite sequences of elements in Σ. It should be clear that such structures can also be modeled by labeled transition systems. This approach is preferable here since it provides a common framework for the logics in the following chapters.

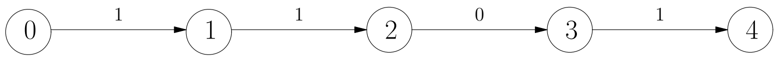
Definition 2.2 An LTS $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ over some set $\mathcal{P}$ of atomic propositions is called a *finite word*, if

- $|\mathcal{S}| < \infty$,
- $\mathcal{S}$ can be linearly ordered as $\{s_0, \dots, s_n\}$ s.t.
  - for all $i = 0, \dots, n-1$ there is exactly one transition from s_i and this is $s_i \xrightarrow{a} s_{i+1}$ for some $a \in \Sigma$,
  - there is no $s \in \mathcal{S}$ with $s_n \xrightarrow{a} s$ for any $a \in \Sigma$,
- $\mathcal{P} = \emptyset$ and $L(s) = \emptyset$ for all $s \in \mathcal{S}$,

We sometimes call s_0 the *starting state* of $\mathcal{T}$.

We will not distinguish formally between the finite word $\mathcal{T}$ (as a transition system) and the finite word obtained as the concatenation of the transition labels between s_0 and s_n.

Example 2.2 The graph below depicts a transition system representing the word 1011 over a binary alphabet $\Sigma = \{0, 1\}$.



Clearly, finite words according to this definition are just another representation of elements of Σ^*. It is sometimes customary to encode finite words of length n as node-labeled linear transition system over a single accessibility relation and n states only. There is no major difference between these two styles, and satisfiability preserving translations exist for modal logics that switch between these two styles [Tho75]. These work equally well for the temporal logics presented here. Here we prefer the encoding as edge-labeled linear structures of length $n+1$ because it simplifies the technicalities in the correspondence results between logics and grammars slightly. It also avoids tedious case distinctions that would become necessary in the presence of the empty word ϵ since the state space of transition systems is usually assumed to be non-empty.

Definition 2.3 A *bisimulation* between two transition systems $\mathcal{T}_1 = (\mathcal{S}_1, \{\xrightarrow{a} \mid a \in \Sigma\}, L_1)$ and $\mathcal{T}_2 = (\mathcal{S}_2, \{\xrightarrow{a} \mid a \in \Sigma\}, L_2)$ – not necessarily distinct – over a set $\mathcal{P}$ of atomic propositions is a binary relation $B \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ s.t. for all $(s,t) \in B$:

- $L_1(s) = L_2(t)$, and
- for all $s' \in \mathcal{S}_1$ with $s \xrightarrow{a} s'$ for some $a \in \Sigma$ there is a $t' \in \mathcal{S}_2$ s.t. $t \xrightarrow{a} t'$ and $(s',t') \in B$, and
- for all $t' \in \mathcal{S}_2$ with $t \xrightarrow{a} t'$ for some $a \in \Sigma$ there is an $s' \in \mathcal{S}_2$ s.t. $s \xrightarrow{a} s'$ and $(s',t') \in B$.

Two states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ are *bisimilar* if there is a bisimulation B with $(s,t) \in B$, written $\mathcal{T}_1, s \sim \mathcal{T}_2, t$.

A logic $\mathcal{L}$ that is interpreted over labeled transition systems is called *bisimulation-invariant* if no formula distinguishes bisimilar structures, i.e. if for all closed $\varphi \in \mathcal{L}$, all LTS $\mathcal{T}_1, \mathcal{T}_2$ with states s,t s.t. $\mathcal{T}_1, s \sim \mathcal{T}_2, t$ we have: $\mathcal{T}_1, s \models \varphi$ iff $\mathcal{T}_2, t \models \varphi$.[1]

It has been argued (and is commonly accepted) that bisimulation is the right notion of behavioural equivalence between models of programs [Mil89]. It is also the strongest in the family of equivalence relations that are considered in the literature and that are designed to capture the notion of equivalence for state-based models of programs [CH93]. We therefore restrict our attention to bisimulation-invariant logics in the following chapters.

[1]The satisfaction relation $\models$ obviously depends on the logic at hand. Here we simply accept that every logic comes with such a relation.

2.2 Properties of Labeled Transition Systems

This thesis is about *properties* – a seemingly vague term that can be defined precisely though. A property of labeled transition systems is simply a subset of the set of all labeled transition systems. Equally, a property of finite words is a set of (encodings of) finite words, i.e. a language. The question of what makes a property regular or not is then, as in formal language theory, a question of what devices are capable of defining the corresponding subset. Such a device may be a *finite automaton*, a *resource-bounded Turing Machine*, a *formal grammar*, a *formula* of a particular *logic*, etc. The set of all finite words *defined* by a finite automaton or Turing Machine is, as usual, simply their *language*. The set of structures *defined* by a logical formula is the set of its models, etc.

Properties of transition systems or finite words can then be grouped into classes according to the type of device needed for their definition, e.g. the class of all *regular properties* as the class of all sets of finite words that can be defined by a finite automaton. We will consider three types of formalisms to define such properties: *formal languages*, *complexity classes*, and *logics*. The former two are very much the same, and the study of the interdependency between the latter two is known as descriptive complexity theory, c.f. Sect. 2.5 below.

Let $\mathcal{K} := \{(\mathcal{T}, s) \mid s \text{ is a state of the LTS } \mathcal{T}\}$. All the logics $\mathcal{L}$ in this work are equipped with a relation $\models \subseteq \mathcal{K} \times \mathcal{L}$ that explains which states of which transition have the property defined by a closed formula of $\mathcal{L}$. It is then possible to compare different logics w.r.t. the properties they can express.

Definition 2.4 Let φ, ψ be two temporal formulas (not necessarily of the same logic). They are *equivalent*, written $\varphi \equiv \psi$ iff for all labeled transition systems $\mathcal{T}$ and all their states s we have $\mathcal{T}, s \models \varphi$ iff $\mathcal{T}, s \models \psi$.

We will also use a logic's name $\mathcal{L}$ to denote the class of all languages (of finite words or transition systems) that can be defined by a formula of this logic, i.e.

$$\mathcal{L} \;:=\; \{(K, s) \in \mathcal{K} \mid \exists \varphi \in \mathcal{L}.\; K, s \models \varphi\}$$

This is not a cyclic definition. It merely abuses notation s.t. the right-hand $\mathcal{L}$ denotes a logic as a syntactical object whereas on the left-hand side $\mathcal{L}$ is a semantical one.

Hence, we can write $\mathcal{L}_1 \subseteq \mathcal{L}_2$, resp. $\mathcal{L}_1 \subsetneq \mathcal{L}_2$ to denote that $\mathcal{L}_2$ is (strictly) more expressive than $\mathcal{L}_1$. Similarly, $\mathcal{L}_1 = \mathcal{L}_2$ states that the logics are equi-expressive.

We will compare different temporal logics w.r.t. their expressive power on arbitrary labeled transition systems as well as finite words. In the latter case the above definition of $\equiv$ is simply restricted to all $\mathcal{T}$ which are finite words according to Definition 2.2. We will not invent a new notation for equi-expressiveness etc. on finite words but rather state explicitly when an expressibility result holds over finite words only.

2.3 The Chomsky Hierarchy

We assume familiarity with the *Chomsky hierarchy* of formal languages [Cho62]. REG, CFL, CSL, and RE denote the classes of *regular*, *context-free*, *context-sensitive*, and *recursively enumerable* languages over some alphabet Σ. Furthermore, let DCSL denote the class of *deterministic context-sensitive languages*, i.e. those that are recognised by a deterministic linear bounded Turing Machine. Finally, let PDA and DPDA be the classes of languages that can be recognised by a *non-deterministic*, resp. *deterministic pushdown automaton*. For detailed definitions see [HU80].

Theorem 2.1
REG $\subsetneq$ DPDA $\subsetneq$ CFL $=$ PDA $\subsetneq$ DCSL $\subseteq$ CSL $\subsetneq$ RE.

2.4 The Modal μ-Calculus

While the Chomsky hierarchy provides a reasonably good framework for the categorisation of properties of finite words, there is no comparable framework for arbitrary transition systems. Instead, we will use Kozen's modal μ-calculus as a reference point against which to measure other logics. This is because it captures various other logics, and it is the counterpart to the class of regular properties of finite words, see Thm. 2.2 below.

Definition 2.5 Let Σ be a finite alphabet, $\mathcal{P}$ be an at most countably infinite set of propositions and $\mathcal{V}$ be a countably infinite set of variable names. Formulas of the *modal μ-calculus* ($\mathcal{L}_\mu$) in positive normal form are given by the following grammar.

$$\varphi \quad ::= \quad q \mid \neg q \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $a \in \Sigma$, $q \in \mathcal{P}$, and $X \in \mathcal{V}$.

We usually write $\Diamond\varphi$ instead of $\bigvee_{a\in\Sigma}\langle a\rangle\varphi$ and $\Box\varphi$ instead of $\bigwedge_{a\in\Sigma}[a]\varphi$. This also applies to other temporal logics with these modal operators defined later on.

Definition 2.6 Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ be an LTS. Formulas of $\mathcal{L}_\mu$ are interpreted over $\mathcal{T}$ in the following way. An environment $\rho : \mathcal{V} \to 2^{\mathcal{S}}$ is used to interpret free variables.

$$\begin{array}{rcl}
[\![\, q \,]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid q \in L(s)\} \\
[\![\, \neg q \,]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid q \notin L(s)\} \\
[\![\, X \,]\!]^{\mathcal{T}}_{\rho} & := & \rho(X) \\
[\![\, \varphi \vee \psi \,]\!]^{\mathcal{T}}_{\rho} & := & [\![\, \varphi \,]\!]^{\mathcal{T}}_{\rho} \cup [\![\, \psi \,]\!]^{\mathcal{T}}_{\rho} \\
[\![\, \varphi \wedge \psi \,]\!]^{\mathcal{T}}_{\rho} & := & [\![\, \varphi \,]\!]^{\mathcal{T}}_{\rho} \cap [\![\, \psi \,]\!]^{\mathcal{T}}_{\rho} \\
[\![\, \langle a\rangle\varphi \,]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid \exists t \in [\![\, \varphi \,]\!]^{\mathcal{T}}_{\rho} \text{ and } s \xrightarrow{a} t\} \\
[\![\, [a]\varphi \,]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : s \xrightarrow{a} t \Rightarrow t \in [\![\, \varphi \,]\!]^{\mathcal{T}}_{\rho}\} \\
[\![\, \mu X.\varphi \,]\!]^{\mathcal{T}}_{\rho} & := & \bigcap\{T \subseteq \mathcal{S} \mid [\![\, \varphi \,]\!]^{\mathcal{T}}_{\rho[X\mapsto T]} \subseteq T\} \\
[\![\, \nu X.\varphi \,]\!]^{\mathcal{T}}_{\rho} & := & \bigcup\{T \subseteq \mathcal{S} \mid T \subseteq [\![\, \varphi \,]\!]^{\mathcal{T}}_{\rho[X\mapsto T]}\}
\end{array}$$

Here, $\rho[X \mapsto T]$ is the function that maps X to T and agrees with ρ on all other arguments.

Example 2.3 Using nestings of least ("finitely often") and greatest fixpoint quantifiers ("infinitely often") it is possible to reason about infinite occurrences in various ways.

"There is a path on which p holds everywhere and q holds infinitely often" is expressed as

$$\nu X.\mu Y.(q \wedge p \wedge \Diamond X) \vee (p \wedge \Diamond Y)$$

and the next formula defines the property "on all paths that satisfy q infinitely often, p holds everywhere."

$$\nu X.\big((p \wedge \Box X) \vee \mu Y.\nu Z.\Box Y \vee (\neg q \wedge \Box Z)\big)$$

Theorem 2.2 [EJ91, JW96, Rab69]
The following are equivalent for a bisimulation-invariant property L of transition systems.

a) L is $\mathcal{L}_\mu$-definable.

b) L is definable by a sentence of Monadic Second-Order Logic.

c) L is definable by a Rabin tree automaton.

It is therefore reasonable to call a property *regular* (upto bisimulation) iff it is definable in the modal μ-calculus. In particular, the following result is a direct consequence of this. We remark that it also holds for ω-words [BKP86, Var88, BB89].

Theorem 2.3
On finite words: $\mathcal{L}_\mu = \text{REG}$.

2.5 Descriptive Complexity

Definition 2.7 For a function $f(n)$ let DTIME($f(n)$), NTIME($f(n)$), DSPACE($f(n)$), and NSPACE($f(n)$) be the class of languages that can be recognised by a deterministic, resp. non-deterministic Turing Machine in time, resp. space $f(n)$. For a class F of functions, let DTIME(F) $:= \bigcup_{f \in F}$ DTIME(f).

Let $2_0^{f(n)} := f(n)$ and $2_{k+1}^{f(n)} := 2^{2_k^{f(n)}}$. Important complexity classes mentioned in the following chapters are, for any $k \in \mathbb{N}$,

$$\begin{array}{rcl} k\text{EXPTIME} & := & \text{DTIME}(\{\ 2_k^{p(n)} \mid p(n) \text{ polynomial }\}) \\ \text{EXPTIME} & := & 1\text{EXPTIME} \\ \text{P} & := & 0\text{EXPTIME} \\ \text{PSPACE} & := & \text{DSPACE}(\{\ p(n) \mid p(n) \text{ polynomial }\}) \end{array}$$

Consider the *model checking* problem for a logic $\mathcal{L}$. Given a $\varphi \in \mathcal{L}$, and an LTS $\mathcal{T}$ with starting state s, decide whether or not $\mathcal{T}, s \models \varphi$ holds. If $\mathcal{T}$ is a finite word, this is simply the word problem for a certain class $\mathcal{L}$ of specifications.

The computational complexity of this problem will also be called the *combined complexity* of $\mathcal{L}$'s model checking problem or simply of $\mathcal{L}$. The term "combined" expresses that the input size of this problem is the sum of the size of the LTS and the size of the formula. This is often too coarse for theoretical considerations because it hides which part of the input is a possible cause of the high complexity. It is also a bad measure for practical purposes.

Verification via model checking may be embedded in a design cycle where erroneous models a subject to alterations, redesign, extensions etc. The correctness properties to be checked, however, often remain unchanged (unless a mistake in the formula occurs for example). We will therefore consider two more special model checking problems when either the LTS or the formula is considered fixed, hence, not part of the input. We will speak of the *data complexity* of $\mathcal{L}$ as the computational complexity of its model checking problem for any fixed formula $\varphi \in \mathcal{L}$. This provides a more realistic measure for the usefulness of model checking for this logic in program verification.

The data complexity also allows us to compare logics over finite words directly to complexity classes. We say that a logic $\mathcal{L}$ *captures* a complexity class $\mathcal{C}$, written $\mathcal{L} = \mathcal{C}$, if

- the data complexity of $\mathcal{L}$ is in $\mathcal{C}$, and
- for every language $L \in \mathcal{C}$ there is a $\varphi_L \in \mathcal{L}$ s.t. $L = \{(\mathcal{T}, s) \mid \mathcal{T}, s \models \varphi\}$

Hence, formulas of $\mathcal{L}$ behave exactly like resource-bounded Turing Machines from $\mathcal{C}$. This opens up possibilities for the transferral of (in-)expressibility results from complexity classes to logics, c.f. Chp. 6 and 7.

We will speak of the *expression complexity* of a logic as the computational complexity of its model checking problem on any fixed LTS. For verification purposes its use may be void. However, it provides insight into the complexity of the logic itself rather than the complexity of the interaction between a logic and labeled transition systems.

2.6 Satisfiability and Validity

The second important decision problem for a logic $\mathcal{L}$ besides model checking is its satisfiability problem. Given a $\varphi \in \mathcal{L}$ decide whether or not there is an LTS $\mathcal{T}$ with starting state s s.t. $\mathcal{T}, s \models \varphi$. This problem will receive less attention than the model checking problem in the following chapters for a simple reason: it is undecidable for all the temporal logics of non-regular properties considered here, whereas model checking remains decidable for all of them on finite transition systems.

Both have simple explanations. Model checking on finite-state systems is trivially decidable by brute-force methods, i.e. enumeration and testing of all the (finitely many) semantical objects. This argument has been used to establish decidability of the model checking problems for the logics in Chp. 5 and 7 for example [MO99, VV04].

For satisfiability remember that – on finite words – model checking is in fact the well-known word or membership problem. Satisfiability then is the equally well-known non-emptiness problem. This is decidable for regular languages and, perhaps surprisingly, for non-regular but context-free languages. However, note that the class of context-free languages is not closed under intersections. A logic corresponding to that class could not contain the intersection operator. Hence, it would be impossible to conjoin program specifications in this logic which would make it useless for verification purposes. Comparing a logic's satisfiability problem to the non-emptiness problem for context-free languages is therefore meaningless. The problem that is more appropriate is the intersection problem for them: given two context-free languages L_1 and L_2, decide whether or not $L_1 \cap L_2 \neq \emptyset$ holds. This, however, is known to be undecidable [GR63]. Hence, we cannot expect to have a sensible but decidable logic that can express context-free properties.

The same holds of course for the validity problem of a logic since it is only the complement of its satisfiability problem provided that the logic is closed under complements. All those considered here are.

2.7 Finitary Axiomatisations and the Finite Model Property

We first provide a very short introduction to the theory of undecidability.

Definition 2.8 Formulas and terms of *First-Order Arithmetic* over a set $\mathcal{V}_1$ of variables are given by the following grammar.

$$\begin{array}{rcl} \varphi & ::= & t = t \mid t < t \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists n.\varphi \\ t & ::= & n \mid x \mid t + t \mid t * t \end{array}$$

where $n \in \mathbb{N}$ and $x \in \mathcal{V}_1$.

These formulas are interpreted over the structure of natural numbers with the usual semantics of equality, less-than, addition and multiplication. We write $n_1, \ldots, n_m \models \varphi(x_1, \ldots, x_m)$ to indicate that the formula φ is true when its free variables $x_1, \ldots, x_m$ are interpreted by $n_1, \ldots, n_m$.

Second-Order Arithmetic extends this logic with variables $X \in \mathcal{V}_2$ for relations between natural numbers and quantification over them.

$$\varphi \quad ::= \quad t = t \mid t < t \mid X(t, \ldots, t) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists n.\varphi \mid \exists X.\varphi$$

$$t \ ::= \ n \mid x \mid t+t \mid t*t$$

where $n \in \mathbb{N}$, $x \in \mathcal{V}_1$, and $X \in \mathcal{V}_2$. Again, their interpretation is standard. Second-order variables are interpreted by relations of matching arity.

A formula $\varphi(x)$ with one free variable defines a subset $L(\varphi)$ of the natural numbers. $L(\varphi) := \{n \mid n \models \varphi(x)\}$. Clearly, this can be seen as a language over the alphabet $\Sigma = \{0,1\}$ for instance.

Definition 2.9 The *arithmetic hierarchy* is based upon the class $\Sigma^0_0 = \Pi^0_0$ of languages defined by a quantifier-free arithmetic formula in the following way.

$$\begin{aligned}
\Sigma^0_{n+1} &:= \{L \mid L = L(\exists x_1 \ldots \exists x_k.\psi) \text{ for some } k \in \mathbb{N} \text{ and } \psi \in \Pi^0_n\} \\
\Pi^0_{n+1} &:= \{L \mid L = L(\forall x_1 \ldots \forall x_k.\psi) \text{ for some } k \in \mathbb{N} \text{ and } \psi \in \Sigma^0_n\}
\end{aligned}$$

The *analytic hierarchy* is based upon the class

$$\Sigma^1_0 \ := \ \bigcup_{n \in \mathbb{N}} \Sigma^0_n \ = \ \bigcup_{n \in \mathbb{N}} \Pi^0_n \ =: \ \Pi^1_0$$

using a similar construction with second-order quantification.

$$\begin{aligned}
\Sigma^1_{n+1} &:= \{L \mid L = L(\exists X_1 \ldots \exists X_k.\psi) \text{ for some } k \in \mathbb{N} \text{ and } \psi \in \Pi^1_n\} \\
\Pi^1_{n+1} &:= \{L \mid L = L(\forall X_1 \ldots \forall X_k.\psi) \text{ for some } k \in \mathbb{N} \text{ and } \psi \in \Sigma^1_n\}
\end{aligned}$$

Theorem 2.4 [Rog67]
For all $n \in \mathbb{N}$ and all $i \in \{0,1\}$:

a) $\Sigma^i_n \subsetneq \Sigma^i_{n+1}$, $\Pi^i_n \subsetneq \Pi^i_{n+1}$,

b) $\Sigma^i_n \subsetneq \Pi^i_{n+1}$, $\Pi^i_n \subsetneq \Sigma^i_{n+1}$.

The levels of the arithmetic and the analytic hierarchy represent different grades of undecidability. Σ^0_1 is exactly the class of semi-decidable languages. Consequently, Π^0_1 is the class of co-semi-decidable languages. Hence, everything that is properly included in Σ^0_2 or Π^0_2 or beyond is not even semi-decidable.

Theorem 2.5 [Rog67]
$\Sigma^0_1 = \text{RE}$.

These degrees of undecidability may not matter for practical purposes: if a problem is undecidable then it cannot be solved automatically regardless of how undecidable it is. However, the different degrees of undecidability of a logic's satisfiability problem do matter in the context of a decidable model checking problem, as we will explain.

Definition 2.10 A logic $\mathcal{L}$ has the *finite model property* if for all satisfiable $\varphi \in \mathcal{L}$ there is an LTS $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ with an $s \in \mathcal{S}$ s.t. $|\mathcal{S}| < \infty$ and $\mathcal{T}, s \models \varphi$.

The following describes a connection between decidability of the model checking and satisfiability checking problems as well as the finite model property.

Theorem 2.6
If a logic $\mathcal{L}$ has a decidable model checking problem and the finite model property then its satisfiability problem is semi-decidable.

This is simply because one can enumerate all possible finite models and check each of them consecutively using the decidability of the model checking problem.

The contrapositive of this theorem holds in general for all of the temporal logics of the following chapters. Their satisfiability problems are usually provably not semi-decidable. Hence, they cannot have the finite model property. This is not too surprising since the finite model property in conjunction with bisimulation-invariance also contradicts the possibility of expressing non-regular properties.

But non-semi-decidability also entails a result about the axiomatisability of a logic $\mathcal{L}$. A finitary axiomatisation is a finite set of axioms and rules with finitely many premisses for every conclusion. A proof for a formula $\varphi \in \mathcal{L}$ is a finite tree whose root is labeled with φ, whose leaves are labeled with axioms and whose internal nodes are instances of the rules. An axiomatisation is sound if every provable formula is valid. It is complete if the converse always holds.

Theorem 2.7
If a logic $\mathcal{L}$ has a complete finitary axiomatisation then its validity problem is semi-decidable.

The proof proceeds very much along the same lines as the argument above for Thm. 2.6. Note that finite proof trees can also be enumerated and checked for being a proof. Completeness ensures that a proof for φ will be among the enumerated ones if φ is valid.

As said above, the logics considered here have satisfiability and, hence, validity problems of high degree of undecidability. Hence, in addition to the lack of finite model property they also do not admit finitary axiomatisations.

Chapter 3

The Space Above Context-Free Languages

3.1 Closures of Language Classes

Definition 3.1 Let $\mathcal{L}$ be some class of formal languages over some alphabet Σ. We consider three types of closures of $\mathcal{L}$.

- The *positive Boolean closure* $\mathbb{B}^+(\mathcal{L})$ is the smallest class of languages that subsumes $\mathcal{L}$ and is closed under finite unions and finite intersections.
- The *Boolean closure* $\mathbb{B}(\mathcal{L})$ is the smallest class of languages that subsumes $\mathcal{L}$ and is closed under finite unions, finite intersections, and complements.
- The *star-free closure* $\mathbb{SF}(\mathcal{L})$ is the smallest class of languages that subsumes $\mathcal{L}$ and is closed under finite unions, finite intersections, complements, and compositions of two languages.

Clearly, for any class $\mathcal{L}$ we have

$$\mathcal{L} \;\subseteq\; \mathbb{B}^+(\mathcal{L}) \;\subseteq\; \mathbb{B}(\mathcal{L}) \;\subseteq\; \mathbb{SF}(\mathcal{L})$$

In many cases, these closures do not increase the underlying class. For example $\mathbb{SF}(\text{REG}) = \text{REG}$. The same holds for context-sensitive languages.

Theorem 3.1 [Imm88, Sze88]
$\mathbb{SF}(\text{CSL}) = \text{CSL}$.

Note that such a result does not hold for $\mathcal{L} = \text{CFL}$. A direct consequence of Thm. 3.1 is then the following.

Theorem 3.2
$\mathbb{SF}(\text{CFL}) \subseteq \text{CSL}$.

3.2 Alternating Context-Free Languages

Definition 3.2 Let Σ be a finite set of action names. A *context-free grammar* (CFG) over Σ is a tuple $G = (N, \Sigma, S, P)$ where

- N is a finite set of non-terminals,
- $S \in N$ is the designated starting symbol,
- $P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of productions.

As usual, we write $X \to w$ instead of $(X, w) \in P$, and use ϵ to denote the empty word.

The language of words generated by G is $L(G) := \{w \mid w \in \Sigma^* \text{ and } S \Rightarrow^* w\}$ where $\Rightarrow$ is the extension of $\to$ to arbitrary sentential forms: for all $X \in N$ and all $u, w, v \in (N \cup \Sigma)^*$ let $uXv \Rightarrow uwv$ if $X \to w$.

Most of the logics considered in the following chapters have the ability to model context-free effects. In addition, they are closed under intersections. Thus, they cannot coincide with CFL on finite words. Note that $\mathbb{B}(\text{CFL})$ and $\mathbb{SF}(\text{CFL})$ only admit a bounded number of intersections to form a language. However, temporal logics usually have recursion operators like fixpoint quantifiers that allow properties to be defined using an unbounded number of intersection operators. This is reflected in the concept of an *alternating grammar* in which non-terminals are either existential – as they are in context-free grammars – or universal. The latter generate a word if *all* possible rule applications lead to this word.

Such grammars were studied by Moriya et al. [Mor89, MHHO05] and Ibarra et al. [IJW92]. However, their model of universal choice is rather strange. It commutes with sequential composition which is an existential construct. As a consequence, the standard construction for the concatenation of two grammars only over-approximates the actual concatenation of their languages.

Definition 3.3 An *alternating context-free grammar* according to Moriya etc. is a tuple $G = (N_\exists, N_\forall, \Sigma, S, P)$ with $N := N_\exists \cup N_\forall$ and the rest as above. We usually write productions like $A \to \alpha_1 \& \ldots \& \alpha_k$ to indicate $A \in N_\forall$.

A *derivation* in G is a tree whose nodes are labeled with sentential forms over $N \cup \Sigma$. It is simply explained extending the derivation relation $\Rightarrow$ in context-free grammars. Each derivation starts with the sentential form S. An existential nonterminal $A \in N_\exists$ is rewritten in the usual style: $uAv \Rightarrow uxv$ if $A \to x$. A universal non-terminal $A \in N_\forall$ rewrites to *all* possibilities. $uAv \Rightarrow ux_1v, \ldots, ux_kv$ if $A \to x_1 \& \ldots \& x_k$. Clearly, these derivations can be arranged to form a tree.

Such a derivation for a word w is called successful if all paths end in w.

Here we only consider left-most derivations which are more restrictive than unconstrained derivations [IJW92, MHHO05]. We write $L(G)$ for the set of words that can be leftmost-derived in G, and ACFL^M_{lm} for the set of languages that can be generated by such a grammar.

Example 3.1 The following example illustrates the strange effect that the interaction between universal choice and concatenation has in this model.

Let $G_i = (N_i, \Sigma, S_i, P_i)$ for $i = 1, 2$ be two context-free grammars. Then the disjoint union of the two with the additional production $S \to S_1 \& S_2$ is an alternating context-free grammar that generates $L(G_1) \cap L(G_2)$. Similarly, their disjoint union with the additional production $S \to S_1 S_2$ is an (even non-alternating) context-free grammar that generates $L(G_1)L(G_2)$.

These standard constructions can generate intersections of concatenations, but they fail to generate concatenations of intersections.

$$S \to ABA \qquad A \to \epsilon \mid a \qquad B \to ab \;\&\; ba$$

This grammar generates the language $\{aba\}$ even though no word can be derived from the non-terminal B.

Theorem 3.3
$\mathbb{B}^+(\mathrm{CFL}) \subseteq \mathrm{ACFL}^M_{lm}$.

This is because $\mathbb{B}^+(\mathrm{CFL})$ consists, by the distributive laws for intersections and unions, of intersections of context-free languages only, and the example above shows how they can be generated.

It is not hard to search the tree of derivations for a given word w and a given grammar G recursively using space $O(|w|^2 \cdot \log |G|)$. Hence, the data complexity of a ACFL^M_{lm} is at most deterministic quadratic space.

Theorem 3.4
$\mathrm{ACFL}^M_{lm} \subseteq \mathrm{DSPACE}(O(n^2))$.

Alternation in context-free grammars has also been studied by Okhotin [Okh01]. He uses the term *conjunctive grammars* for what is a context-free grammar with a natural extension to universal choice. A derivation in such a grammar is a linear sequence of sentential forms over an extended alphabet. Additional symbols are used to denote parts which result from a universal branching.

Definition 3.4 A *conjunctive grammar* is a tuple $G = (N_\exists, N_\forall, \Sigma, S, P)$ as above. A *derivation* in G is a finite sequence of sentential forms over $N \cup \Sigma \cup \{(,), \&\}$ assuming that neither of the additional symbols occurs in $N \cup \Sigma$. It starts in S and continues according to the following rules.

$$\begin{aligned} uAv &\Rightarrow uxv \quad \text{if} \quad A \in N_\exists \text{ and } A \to x \\ uAv &\Rightarrow u(x_1\& \dots \&x_k)v \quad \text{if} \quad A \in N_\forall \text{ and } A \to x_1\& \dots \&x_k \\ u(w\& \dots \&w)v &\Rightarrow uwv \end{aligned}$$

It is successful if it ends in in a terminal string w. Again, $L(G) = \{w \mid S \Rightarrow^* w\}$.

The notion of conjunctive grammar does not easily extend to ω-words since the information about which non-terminal has been generated by which is lost in their derivations. A third type of alternating context-free grammar has been introduced in [Lan02a] making up for this deficiency. Since we are not particularly concerned with ω-words we only mention this as a motivation for yet another type of alternating context-free grammar. Derivations in this third formalism are trees. It is therefore easy to explain what a derivation of an ω-word is – using a parity condition on non-terminals for instance.

Definition 3.5 An *alternating context-free grammar* is tuple $G = (N_\exists, N_\forall, \Sigma, S, P)$ with $N := N_\exists \cup N_\forall$ the set of non-terminals and the rest as usual.

A *derivation* for a word $w \in \Sigma^*$ in G is a tree whose nodes are labeled with subwords of w and sentential forms occurring in G. The root is labeled with (w, S), and the tree is built top-down according to the following rules.

- If a node is labeled (v, X) for some $X \in N_\exists$ then there is a unique son labeled (v, u) for some u s.t. $X \to u \in P$.
- If a node is labeled (v, X) for some $X \in N_\forall$ then for every $X \to u \in P$ there is a son labeled (v, u). Furthermore, every son of this node is of this kind.
- If a node is labeled (v, uz) with $u, z \in (N \cup \Sigma)^+$ then there are two sons labeled (v_1, u) and (v_2, z) s.t. $v = v_1 v_2$.

A derivation is successful if every leaf is of the form (a, a) for some $a \in \Sigma$. Let $L(G)$ be the set of words for which there is a successful derivation in G. We write ACFL to denote the class of languages that can be generated by an alternating context-free grammar.

Two reasonably straight-forward inductions – on the height of derivation trees in alternating context-free grammars and on the length of a derivation in a conjunctive grammar – show that not only are the classes of languages generated by these two formalisms the same. A grammar even generates the same language when regarded as a conjunctive or an alternating context-free one. We will therefore not distinguish between these two formalisms anymore and write ACFL for the class of languages that are generated by either of these grammars.

Clearly, ACFL subsumes CFL. Also, the intersection of two languages can easily be modeled using universal choice. This leads to the following result.

Theorem 3.5 [Okh01, Wot73]
$\mathbb{B}^+(\text{CFL}) \subsetneq \text{ACFL}$.

The strictness of this inclusion is witnessed by the language $\{wcw \mid w \in \{a, b\}^*\}$. It can be generated by a conjunctive grammar, i.e. it is alternating context-free [Okh01]. On the other hand it was shown that it cannot be represented as the intersection of finitely many context-free languages [Wot73].

Another extension of context-free grammars are so-called *Boolean grammars* [Okh03a]. In addition to the universal choices from alternating context-free grammars they also contain negation operators that allow to specify what a non-terminal does *not* generate. It should be clear that this imposes additional problems in explaining what the language generated by such a grammar is. For details we refer to [Okh04] and [KNR06]. Here we simple accept it as an extension of context-free grammars.

We use the term BCFL to denote the class of languages that are generated by boolean grammars – the boolean context-free languages. The following is immediate even from the short description of boolean grammars.

Theorem 3.6
$\mathbb{SF}(\text{CFL}) \cup \text{ACFL} \subseteq \text{BCFL}$.

A straight-forward extension of the well-known Cocke-Younger-Kasami algorithm for membership in a context-free language [You67] yields an upper bound on BCFL and, hence, ACFL: every boolean / alternating context-free language can be recognised in deterministic quadratic space. This is because only a table of size $O(n^2)$ needs to be filled when n is the length of the input word.

However, a better algorithm for the word problem (with a fixed grammar) of boolean grammars runs in deterministic linear space [Okh04]. Hence, every boolean context-free language is also a deterministic context-sensitive one.

Theorem 3.7 [Okh04]
$\text{BCFL} \subseteq \text{DSPACE}(O(n))$.

It is not known whether or not this inclusion is strict.

Another obvious choice for the extension of context-free languages with alternation is the correspondence between CFL and PDA. An *alternating push-down automaton* [LLS84] has, in addition to non-deterministic choice, universal choices as well. In a configuration with a universal state it recognises a word if all possible successor configurations lead to accepting configurations. Let APDA denote that class of languages that are recognisable be such an automaton. Clearly, APDA contains PDA, and this inclusion is strict since PDA is not closed under intersections which APDA trivially is.

Theorem 3.8 [LLS84, CKS81]
$\text{APDA} = \text{DTIME}(2^{O(n)})$.

It is tempting to assume that alternating pushdown automata recognise exactly those languages that are generated by alternating context-free grammars – lifting the well-known correspondence between these grammars and automata from the non-deterministic model to the alternating one. This is not true even though there is a published claim about this by Moriya [Mor89]. In fact, both directions fail in the general case.

- In the languages generated by alternating context-free grammars not every word has a left-most derivation which is a crucial ingredient for the proof of the simulation of context-free grammars by pushdown automata.
- It is too naïve to simply adapt the standard translation from a push-down automaton to a context-free grammar for the alternating case. Remember that the grammar's non-terminals are triples of states, stack symbols and states. The second state is used by the grammar to "guess" which state the automaton will be in after removing the currently pushed symbol onto the stack. This fails for an alternating push-down automaton due to universal branching. There is no unique such state but different branches can have led to different states once this symbol is popped.

The first problem can be avoided by only considering left-most derivations as done above in the case of ACFL^M_{lm} for instance. In order to overcome the second problem, Ibarra et. al [IJW92] define so called *linear-erasing alternating context-free grammars* and show that they generate exactly those languages that are recognisable in deterministic time $2^{O(n)}$ which coincides with APDA according to Thm. 3.8.

3.3 A General Inexpressivity Theorem

Definition 3.6 Let $G = (N, \Sigma, S, P) \in \mathcal{G}$ be a formal grammar. W.l.o.g. we can assume $N = \{A_1, \dots, A_n\}$ for some $n \in \mathbb{N}$ and $\Sigma = \{a_1, \dots, a_m\}$ for some $m \in \mathbb{N}$. Note that every such grammar is uniquely determined by the list P of its productions. We introduce the convention that the first non-terminal listed in P is the starting symbol, and that every non-terminal listed on the right-hand side of a production but not anywhere on the left-hand side generates the empty language. This can be modeled by $A \to aA$ for example.

It should be clear that every such grammar G can be represented as a word $\widetilde{G}$ over the alphabet $\Sigma_0 := \{A, a, \to, |, \&, ;, (,), 0, 1\}$.

We call a class $\mathcal{G}$ of formal grammars *auto-presentable* if $L^{\mathcal{G}}_{present} := \{\widetilde{G} \mid G \in \mathcal{G}\}$ is itself a language that can be defined by a grammar $G \in \mathcal{G}$.

We will write $L(\mathcal{G})$ for the class of languages that is generated by grammars from $\mathcal{G}$.

Note that all classes of grammars from REG up to BCFL are all auto-presentable. In fact, even $L^{\text{BCFL}}_{\mathit{present}} \in \text{REG}$. The classes of context-sensitive grammars are also auto-presentable since string length comparison can be done in linear space.

Definition 3.7 Let $\mathcal{G}$ be a class of auto-presentable grammars and $L^{\mathcal{G}}_{\mathit{diag}} := \{\widetilde{G} \mid G \in \mathcal{G} \text{ and } \widetilde{G} \notin L(G)\}$ be the set of all all representations of such grammars that do not generate themselves.

Note that $L^{\mathcal{G}}_{\mathit{diag}}$ is neither the empty language nor the universal language for any $\mathcal{G}$ that contains grammars for the empty and the universal language.

Theorem 3.9
Let $\mathcal{G}$ be any class of auto-presentable grammars, and $\mathcal{C}$ be a complexity class s.t.

1. the combined model checking problem for $\mathcal{G}$ is in $\mathcal{C}$, and
2. co$-\mathcal{C}$ is closed under compositions and closed under intersections with $L(\mathcal{G})$, and
3. the pairing mapping $f(w) = (w, w)$ is computable in co$-\mathcal{C}$.

Then co$-\mathcal{C} \not\subseteq L(\mathcal{G})$.

Proof Let $\mathcal{G}$ be fixed. We need to present a language in co$-\mathcal{C}$ that is not included in $L(\mathcal{G})$. Such a language is $L^{\mathcal{G}}_{\mathit{diag}}$.

First of all observe that by assumption (1) the set $\{(\tilde{G}, w) \mid w \in L(G)\}$ is in $\mathcal{C}$. By (2) co$-\mathcal{C}$ is closed under intersection with any language from $L(\mathcal{G})$, in particular $L^{\mathcal{G}}_{\mathit{present}}$. So we have $\{(\tilde{G}, w) \mid w \notin L(G)\} \in$ co$-\mathcal{C}$. By (2) and (3) we also have $L^{\mathcal{G}}_{\mathit{diag}} \in$ co$-\mathcal{C}$.

On the other hand, suppose that $L^{\mathcal{G}}_{\mathit{diag}} \in L(\mathcal{G})$. Then there is a formal grammar $G_{\mathit{diag}} \in \mathcal{G}$ s.t. $L^{\mathcal{G}}_{\mathit{diag}} = L(G_{\mathit{diag}})$. Now consider its word representation $\widetilde{G_{\mathit{diag}}}$. We have

$$\widetilde{G_{\mathit{diag}}} \in L^{\mathcal{G}}_{\mathit{diag}} \quad \text{iff} \quad \widetilde{G_{\mathit{diag}}} \notin L(G_{\mathit{diag}}) \quad \text{iff} \quad \widetilde{G_{\mathit{diag}}} \notin L^{\mathcal{G}}_{\mathit{diag}}$$

which is obviously a contradiction. ■

Note that if $\mathcal{C}$ is a deterministic complexity class we have co$-\mathcal{C} = \mathcal{C}$. Also, the combined model checking complexity of a specification formalism is an upper bound on its data complexity. In that case the conclusion of Thm. 3.9 is equivalent to $L(\mathcal{G}) \subsetneq \mathcal{C}$.

Unfortunately, the currently best known upper bounds on the combined model checking complexity of various formal grammars are too high to yield separation results that are stronger than those we get out of the space hierarchy theorem. For example, the best known upper bound on the combined complexity of ACFL is DSPACE($O(n^3)$). Hence, we get ACFL $\subsetneq$ DSPACE($O(n^3)$), while the space hierarchy theorem already yields ACFL $\subsetneq$ DSPACE($f(n)$) for any space-constructible function f s.t. $n = o(f(n))$.

As mentioned above, the combined complexity for ACFL$^M_{lm}$ is deterministic space $O(n^2 \log n)$, thus we have ACFL$^M_{lm} \subsetneq$ DSPACE($O(n^2 \log n)$) which is also a consequence of the space hierarchy theorem.

Chapter 4

Propositional Dynamic Logic of Context-Free Programs

4.1 Syntax and Semantics

Definition 4.1 Let Σ be a finite set of action names, $\mathcal{P}$ be an at most countably infinite set of atomic propositions. Formulas of *Propositional Dynamic Logic of Context-Free Programs* over Σ and $\mathcal{P}$ are given by the following grammar.

$$\varphi \ ::= \ q \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle G \rangle \varphi$$

where $q \in \mathcal{P}$, and $G = (N, \Sigma, S, P)$ is a context-free grammar over Σ.

We will use the standard abbreviations $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $[G]\varphi := \neg\langle G\rangle\neg\varphi$, $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathcal{P}$, and $\mathtt{ff} := \neg\mathtt{tt}$.

Definition 4.2 Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ be an LTS. The transition relations $\xrightarrow{a} \subseteq \mathcal{S} \times \mathcal{S}$ extend to finite words and languages over Σ in a straight-forward way. For all $s, t \in \mathcal{S}$, all $a \in \Sigma$, all $w \in \Sigma^*$, and all $L \subseteq \Sigma^*$ we define

$$\begin{array}{rcl} s \xrightarrow{\epsilon} t & \text{iff} & s = t \\ s \xrightarrow{aw} t & \text{iff} & \exists u \in \mathcal{S}, \text{ s.t. } s \xrightarrow{a} u \text{ and } u \xrightarrow{w} t \\ s \xrightarrow{L} t & \text{iff} & \exists w \in L, \text{ s.t. } s \xrightarrow{w} t \end{array}$$

Given such an LTS and a PDL[CFG] formula φ we explain its satisfaction in $\mathcal{T}$ inductively.

$$\mathcal{T}, s \models q \qquad \text{iff} \qquad q \in L(s)$$

$$\begin{array}{rcl} \mathcal{T}, s \models \varphi \vee \psi & \text{iff} & \mathcal{T}, s \models \varphi \text{ or } \mathcal{T}, s \models \varphi \\ \mathcal{T}, s \models \neg\varphi & \text{iff} & \mathcal{T}, s \not\models \varphi \\ \mathcal{T}, s \models \langle G \rangle \varphi & \text{iff} & \exists t \in \mathcal{S}, \text{ s.t. } s \xrightarrow{L(G)} t \text{ and } \mathcal{T}, t \models \varphi \end{array}$$

Example 4.1
Consider the context-free grammar G over the alphabet $\Sigma = \{a, b\}$ given by the production rule

$$S \rightarrow b \mid aSS$$

Note that $L(G) = \{w \mid \forall u, v \in \Sigma^* : w = uv \Rightarrow |u|_a \leq |u|_b\}$. This is easily seen to be a non-regular language. Moreover, $L(G)$ describes the undesired runs of a buffer if, for example, $b =$ `out` and $a =$ `in`. Hence, the PDL[CFG] formula $[G]\mathtt{ff}$ specifies that on all paths in a transition system the number of b-transitions never exceeds the number of a-transitions, c.f. Ex. 1.1.

Example 4.2
Consider the language $L = \{ww \mid w \in \Sigma^+\}$. This is non-regular, not even context-free. However, it is known that its complement $\bar{L}$ is context-free. Let $\Sigma = \{a, b\}$ for example. Then $\bar{L}$ is generated by the grammar G with production rules

$$\begin{array}{rcl} S & \rightarrow & T_a T_b \mid T_b T_a \\ T_a & \rightarrow & a \mid U T_a U \\ T_b & \rightarrow & b \mid U T_b U \\ U & \rightarrow & a \mid b \end{array}$$

Consequently, the PDL[CFG] formula $[G]\mathtt{ff}$ asserts that all runs of an LTS are of the form $ww\ldots$ for some $w \in \Sigma^+$.

4.2 Expressive Power

Definition 4.3 A right-linear grammar is a CFG $G = (N, \Sigma, S, P)$ s.t. for all $(X, w) \in P$: $w \in \Sigma^*(N \cup \{\epsilon\})$.

Propositional Dynamic Logic (PDL) is obtained from PDL[CFG] by restricting the use of context-free grammars G in a modal operator $\langle G \rangle$ to right-linear grammars only.

Note that usually the modal operators in PDL are parametrised by regular expressions over Σ. The form using right-linear grammars is clearly equi-expressive. But notice that it may not be equi-succinct because of possible exponential blow-ups occurring in the translations from regular expressions to right-linear grammars via nondeterministic finite automata and back – depending on how their sizes are measured.

It is well-known that PDL is a strict fragment of the modal μ-calculus. In fact, there is a very simple and uniform translation of PDL into $\mathcal{L}_\mu$ [Koz83]. The resulting formulas are even alternation-free.

Theorem 4.1
PDL $\subsetneq$ PDL[CFG].

The inclusion holds trivially. The fact that it is strict is witnessed for instance by the formula $\varphi := \langle G \rangle \mathtt{tt}$ where G generates the context-free language $\{a^n b^n \mid n \in \mathbb{N}\}$. It is a simple exercise to show that this property cannot even be expressed by a finite Büchi tree automaton.

Theorem 4.2
On finite words: $\mathbb{B}$(CFL) $\subseteq$ PDL[CFG] $\subseteq$ $\mathbb{SF}$(CFL).

The first "$\subseteq$" is straight-forward. On finite words, a context-free language L is expressed as $\langle G \rangle \mathtt{tt}$ where $L(G) = L$. Furthermore, PDL[CFG] has all boolean operators.

The second "$\subseteq$" is easily shown by induction on the formula structure. Note that on finite words $\langle G \rangle \varphi$ corresponds to left-concatenation with the context-free language $L(G)$. However, $\mathbb{SF}$(CFL) allows left-concatenation with arbitrary languages from $\mathbb{SF}$(CFL). Hence, this inclusion is likely to be strict.

4.3 Complexity Results

Clearly, PDL[CFG] is much more expressive than PDL. This increase is, however, not reflected in the complexity of its model checking problem.

Theorem 4.3 [Lan05]
The model checking problem (combined complexity) for PDL[CFG] is P-complete.

Note that the model checking problem for mono-modal logic K is already P-hard. Inclusion in P is a consequence of the fact that the extended transition relations $\xrightarrow{L(G)}$, where G is a context-free grammar, can be computed in a simple fixpoint iteration. This only needs polynomial time and can be embedded in a model checking procedure for multi-modal logic K. The resulting algorithm has worst-case running time of $O(m^2 \cdot n^5)$ where n is the number of states of the underlying transition system and m is the size of the formula.

Theorem 4.4
The model checking problem on finite words (combined complexity) for PDL[CFG] is P-complete.

This low complexity bound becomes less surprising if one considers the apparent relationship between PDL[CFG]'s model checking problem and the word problem for context-free grammars. Remember that the latter can be solved in time $O(m \cdot n^3)$ where m is the size of a grammar and n is the size of the underlying word [You67].

This relationship breaks apart when considering PDL[CFG]'s satisfiability problem. Note that deciding satisfiability of a formula of the form $\langle G \rangle \mathtt{tt}$ is equivalent to checking $L(G)$ for emptiness which is decidable. However, PDL[CFG] also features conjunctions in the formula, and remember that the intersection problem for two context-free languages is undecidable [GR63]. It is therefore not surprising that the satisfiability problem for PDL[CFG] is also undecidable. Maybe surprising is its degree of undecidability – it is not even in the arithmetic hierarchy.

The upper bound is trivially inherited from Thm. 4.3. The lower bound is a consequence of Thm. 4.2 and the fact that the word problem for context-free languages is P-hard.

Theorem 4.5 [HPS83]
The satisfiability problem for PDL[CFG] is Σ^1_1-complete.

Definition 4.4 Let G be a context-free grammar. With PDL[G] we denote the fragment of PDL[CFG] in which every occurring context-free grammar is either G itself or right-linear.

For better readability we will use symbolic names, e.g. PDL[$a^n b^n$] denotes PDL[G] for some context-free grammar G that generates the well-known non-regular language $\{a^n b^n \mid n \in \mathbb{N}\}$. Harel, Pnueli and Stavi showed that the satisfiability problem for PDL[$a^n b^n$] is already Σ^1_1-complete.

Chapter 5

Fixpoint Logic with Chop

5.1 Syntax and Semantics

Definition 5.1 Let $\Sigma = \{a, b, \ldots\}$ be a finite set of action names, $\mathcal{P} = \{p, q, \ldots\}$ be an at most countably infinite set of atomic propositions and $\mathcal{V} = \{X, Y, \ldots\}$ be a countably infinite set variables. Formulas of *Fixpoint Logic with Chop* (FLC) in positive normal form over Σ and $\mathcal{P}$ are given by the following grammar.

$$\varphi \;::=\; q \mid \neg q \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \mid [a] \mid \varphi ; \varphi \mid \tau \mid X \mid \mu X.\varphi \mid \nu X.\varphi$$

where $a \in \Sigma$, $q \in \mathcal{P}$, and $X \in \mathcal{V}$.

We use the usual abbreviations $\mathtt{tt} := q \vee \neg q$ and $\mathtt{ff} := q \wedge \neg q$ for some $q \in \mathcal{P}$, as well as $\Diamond := \bigvee_{a \in \Sigma} \langle a \rangle$ and $\Box := \bigwedge_{a \in \Sigma} [a]$ etc. We even allow restricted implications of the form $q \to \varphi$ as an abbreviation for $\neg q \vee \varphi$.

Definition 5.2 Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ be an LTS. In order to give the sequential composition operator a natural meaning, the semantics of an FLC formula is a *predicate transformer* of type $2^{\mathcal{S}} \to 2^{\mathcal{S}}$ rather than a predicate of type $2^{\mathcal{S}}$ as in the case of the modal μ-calculus.

Let $2^{\mathcal{S}} \to_{mon} 2^{\mathcal{S}}$ be the set of monotone (w.r.t. $\subseteq$) functions of type $2^{\mathcal{S}} \to 2^{\mathcal{S}}$. This set, together with the partial order $\sqsubseteq$ defined by

$$f \sqsubseteq g \qquad \text{iff} \qquad \forall T \subseteq \mathcal{S} : f(T) \subseteq g(T)$$

forms a complete lattice with joins $\sqcup$ and meets $\sqcap$. Note that this lattice includes all constant functions and is closed under function composition $\circ$. According to the Knaster-Tarski-Theorem [Tar55], least and greatest fixpoints of monotone (w.r.t. $\sqsubseteq$) functionals of type $(2^{\mathcal{S}} \to 2^{\mathcal{S}}) \to (2^{\mathcal{S}} \to 2^{\mathcal{S}})$ exist in this lattice.

These closure properties are used to explain the semantics of FLC over $\mathcal{T}$. It is done inductively using environments $\rho : \mathcal{V} \to (2^{\mathcal{S}} \to_{mon} 2^{\mathcal{S}})$ that interpret free variables in a formula. With $\rho[X \mapsto f]$ we denote the function that maps X to f and behaves like ρ on all other arguments.

$$\begin{aligned}
[\![q]\!]^{\mathcal{T}}_{\rho} &:= \lambda\_.\{s \in \mathcal{S} \mid q \in L(s)\} \\
[\![\neg q]\!]^{\mathcal{T}}_{\rho} &:= \lambda\_.\{s \in \mathcal{S} \mid q \notin L(s)\} \\
[\![\varphi \vee \psi]\!]^{\mathcal{T}}_{\rho} &:= [\![\varphi]\!]^{\mathcal{T}}_{\rho} \sqcup [\![\psi]\!]^{\mathcal{T}}_{\rho} \\
[\![\varphi \wedge \psi]\!]^{\mathcal{T}}_{\rho} &:= [\![\varphi]\!]^{\mathcal{T}}_{\rho} \sqcap [\![\psi]\!]^{\mathcal{T}}_{\rho} \\
[\![\langle a \rangle]\!]^{\mathcal{T}}_{\rho} &:= \lambda T.\{s \in \mathcal{S} \mid \exists t \in \mathcal{S} \text{ s.t. } s \xrightarrow{a} t \text{ and } t \in T\} \\
[\![[a]]\!]^{\mathcal{T}}_{\rho} &:= \lambda T.\{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : s \xrightarrow{a} t \text{ implies } t \in T\} \\
[\![\varphi ; \psi]\!]^{\mathcal{T}}_{\rho} &:= [\![\varphi]\!]^{\mathcal{T}}_{\rho} \circ [\![\psi]\!]^{\mathcal{T}}_{\rho} \\
[\![\tau]\!]^{\mathcal{T}}_{\rho} &:= \lambda T.T \\
[\![X]\!]^{\mathcal{T}}_{\rho} &:= \rho(X) \\
[\![\mu X.\varphi]\!]^{\mathcal{T}}_{\rho} &:= \bigsqcap\{f : 2^{\mathcal{S}} \to_{mon} 2^{\mathcal{S}} \mid [\![\varphi]\!]^{\mathcal{T}}_{\rho[X \mapsto f]} \sqsubseteq f\} \\
[\![\nu X.\varphi]\!]^{\mathcal{T}}_{\rho} &:= \bigsqcup\{f : 2^{\mathcal{S}} \to_{mon} 2^{\mathcal{S}} \mid f \sqsubseteq [\![\varphi]\!]^{\mathcal{T}}_{\rho[X \mapsto f]}\}
\end{aligned}$$

We then define $\mathcal{T}, s \models_{\rho} \varphi$ if $s \in [\![\varphi]\!]^{\mathcal{T}}_{\rho}(\mathcal{S})$.

This induces two different equivalences in FLC. Two formulas φ and ψ are *strongly equivalent*, written $\varphi \equiv \psi$, if for all $\mathcal{T}$ and all ρ: $[\![\varphi]\!]^{\mathcal{T}}_{\rho} = [\![\psi]\!]^{\mathcal{T}}_{\rho}$. Thus, being strongly equivalent means expressing the same property in terms of predicate transformers.

Two formulas φ and ψ are *weakly equivalent*, written $\varphi \approx \psi$, if for all $\mathcal{T}$, all states s and all ρ: $\mathcal{T}, s \models_{\rho} \varphi$ iff $\mathcal{T}, s \models_{\rho} \psi$. Thus, weakly equivalent formulas always define the same set of states in an LTS.

It should be clear that strong equivalence is really stronger than weak equivalence: $\varphi \equiv \psi$ implies $\varphi \approx \psi$. The converse does not hold. For example, $\langle a \rangle \approx \langle a \rangle ; \mathtt{tt}$ but $\langle a \rangle \not\equiv \langle a \rangle ; \mathtt{tt}$.

Note that strong equivalence is the natural notion whereas weak equivalence is the important one for program verification. Two FLC specifications should be regarded as equivalent if the corresponding formulas are weakly equivalent, i.e. if they are satisfied by exactly the same models. Furthermore, FLC is closed under complementation w.r.t. weak equivalence but not w.r.t. strong equivalence [Lan06a].

Example 5.1

Using modal operators "behind" variables it is possible to achieve context-free effects. For example,

$$\varphi_{a,b} \;:=\; \mu X.([a] \wedge \Diamond);([b] \wedge \Diamond) \vee ([a] \wedge \Diamond);X;([b] \wedge \Diamond)$$

says that all paths start with a prefix of the form $a^n b^n$ for some $n \geq 1$. Using conjunctions it becomes possible to define context-sensitive properties. Let $\psi_c := \mu Y.([c] \wedge \Diamond) \vee ([c] \wedge \Diamond);Y$.

$$\varphi \;:=\; \varphi_{a,b};\psi_c \wedge \psi_a;\varphi_{b,c}$$

says that all paths start with a prefix of the form $a^n b^n c^n$ for some $n \geq 1$.

Example 5.2

It is well-known that the property "on all paths eventually q holds" is regular, it is for instance expressed by the CTL formula $\texttt{EF}q$ expresses. In predicate logic this is of the form "$\forall$ paths $\exists$ moment …". If the quantifications are swapped ("$\exists$ moment $\forall$ paths …) then this property – called *uniform inevitability* – is not regular anymore [Eme87]. However, it is expressed by the simple FLC formula

$$\varphi \;:=\; \big(\mu X.\tau \vee X;\Box\big);q$$

Some guidance in reading and understanding such formulas is provided by a game-theoretic characterisation of FLC's model checking problem [Lan02b, Lan06a].

Example 5.3 [LS02]

The following formula shows that FLC is capable of doing unlimited counting.

$$\varphi \;:=\; \nu X.[b];\texttt{ff} \wedge [a];\big(\nu Y.[b] \wedge [a];Y;Y\big);X$$

says that along any path the number of b-actions never exceeds the number of a-actions, c.f. Ex. 1.1 and 4.1.

Example 5.4

Consider the well-known context-sensitive (and non-context-free) language $\{ww \mid w \in \Sigma^*\}$. Its complement is context-free though. It is therefore not surprising that the property "all paths begin with a prefix of the form ww for some $w \in \Sigma^{+}$" is FLC-definable. Note that this would be trivial if $w \in \Sigma^*$ was allowed.

$$\varphi \ := \bigwedge_{a,b\in\Sigma, a\neq b} \big(\nu X.[a] \wedge \Box; X; \Box\big) \ ; \ \big(\nu X.[b] \wedge \Box; X; \Box\big) \ ; \ \mathtt{ff}$$

Compare this to a grammar for the context-free language $\{\Sigma^n a \Sigma^n \Sigma^m b \Sigma^m \mid n, m \in \mathbb{N}\}$ which is exactly the aforementioned complement language.

Example 5.5

Given two non-deterministic, finite automata $\mathcal{A}$ and $\mathcal{B}$ over the alphabet $\Sigma = \{a, b\}$. Let Q_A, Q_B be their resp. sets of states, s_A and s_B the starting and F_A and F_B the final states. Consider the transition system $\mathcal{T}_{\mathcal{A},\mathcal{B}}$ that consists of the disjunct union of $\mathcal{A}$ and $\mathcal{B}$ as graphs plus a new state s with transitions $s \xrightarrow{a} s_A$ and $s \xrightarrow{b} s_B$. Furthermore, the labeling on the states is as follows.

$$\begin{aligned} L(s) &= \emptyset && \\ L(q) &= \{e\} && \text{if } q \in Q_A \setminus F_A \\ L(q) &= \{e, f\} && \text{if } q \in F_A \\ L(q) &= \{u, f\} && \text{if } q \in Q_B \setminus F_B \\ L(q) &= \{u\} && \text{if } q \in F_B \end{aligned}$$

The goal is to show that FLC can express language (non-)inclusion between nondeterministic finite automata. I.e. the aim is to construct a formula φ s.t.

$$\mathcal{T}_{\mathcal{A},\mathcal{B}}, s \models \varphi \qquad \text{iff} \qquad L(\mathcal{A}) \not\subseteq L(\mathcal{B})$$

Note that $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$ iff there is a path emerging from s with labeling aw for some $w \in \Sigma^*$ that ends in a state labeled f, s.t. all paths with labeling bw end in such a state, too.

We define auxiliary formulas $\psi_a := (e \rightarrow \langle a\rangle) \wedge (u \rightarrow [a])$ and equally ψ_b with b instead of a. Then the desired formula is

$$\varphi \ := \ \big(\mu X.(\langle a\rangle \wedge \langle b\rangle) \ \vee \ X; \psi_a \ \vee \ X; \psi_b \ \big); f$$

Again, for a some guidelines about how to read and understand FLC formulas with refer to its game-theory [Lan02b, Lan06a].

5.2 Expressive Power

Theorem 5.1 [MO99]
$\mathcal{L}_\mu \subsetneq$ FLC.

The inclusion is shown by a simple uniform translation. The fact that there are non-regular and FLC-definable properties is also easy to see, c.f. the examples above.

As in the case of the modal μ-calculus one can consider fragments of FLC of bounded alternation depth.

Definition 5.3 Let $\varphi \in$ FLC. Take two variables X, Y in φ and assume they are bound by $\sigma X.\psi$ and $\sigma' Y.\psi'$. We write $X \prec_\varphi Y$ if there is a free occurrence of Y in $\sigma X.\psi$, and use $<_\varphi$ to denote the transitive closure of $\prec_\varphi$.

The *alternation depth* of an FLC formula φ, $ad(\varphi)$ is the number n in a maximally long chain of variables

$$X_0 \;<_\varphi\; X_1 \;<_\varphi\; \ldots \;<_\varphi X_n$$

s.t. for all $i = 0, \ldots, n-1$: X_i and X_{i+1} are not of the same fixpoint type.

Let $\text{FLC}^k := \{\varphi \in \text{FLC} \mid ad(\varphi) \le k\}$. Clearly, $\text{FLC} = \bigcup_{k\in\mathbb{N}} \text{FLC}^k$.

As mentioned earlier, we will also use FLC^k semantically to denote the class of languages of FLC^k formulas.

Theorem 5.2 [Lan06a]
For all $k \in \mathbb{N}$: $\text{FLC}^k \subsetneq \text{FLC}^{k+1}$.

The proof of this result proceeds along the lines of Arnold's argument showing that the alternation hierarchy in $\mathcal{L}_\mu$ is strict over the class of binary trees [Arn99]. It develops a game-theoretic characterisation of FLC's model checking problem and reveal hard formulas for every level of the alternation hierarchy. These express the existence of a winning strategy for one of the player and are, not surprisingly, extensions of the Walukiewicz formulas that do this for the modal μ-calculus. It then uses Banach's fixpoint theorem in a diagonalisation argument to show that these formulas cannot be expressed with less alternation.

Definition 5.4 An FLC formula is *disjunctive* if it can be derived in the following restricted grammar.

$$\varphi^d \ ::= \ \langle a\rangle \mid \varphi^d \vee \varphi^d \mid \varphi^d ; \varphi^d \mid \tau \mid X \mid \mu X.\varphi^d$$

where $a \in \Sigma$ and $q \in \mathcal{P}$. Similarly, *conjunctive* formulas are exactly those that can be derived in the following one.

$$\varphi^c \ ::= \ [a] \mid \varphi^c \wedge \varphi^c \mid \varphi^c ; \varphi^c \mid \tau \mid X \mid \nu X.\varphi^c$$

Formulas of *Heterogenous FLC* (HFLC) over Σ and $\mathcal{P}$ are given by the following grammar.

$$\varphi \ ::= \ q \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi^d ; \varphi \mid \varphi^c ; \varphi$$

where $q \in \mathcal{P}$, and φ^d, resp. φ^c, is a closed and disjunctive, resp. conjunctive formula.

Theorem 5.3
HFLC $\subsetneq$ FLC^0.

The inclusion is not hard to see. Note that μ-quantifiers can only occur in disjunctive parts, and ν-quantifiers can only occur in conjunctive parts. However, these parts have to be closed when put together in an HFLC formula. Hence, the result is always alternation-free.

The strictness of this inclusion becomes apparent when considering the next result. For instance, HFLC is not capable of expressing the (even regular) property "on all paths eventually q".

Theorem 5.4 [LS06]
HFLC = PDL[CFG].

The direction "$\supseteq$" is not difficult. Every modal operator $\langle G\rangle$ for a context-free grammar G can be translated straight-forwardly into a disjunctive formula φ^d_G whose semantics coincides with the relation $\xrightarrow{L(G)}$. Consequently, every $[G]$ can be translated into a conjunctive part. The rest of the translation is uniform.

For the direction "$\subseteq$" one needs to observe that every disjunctive or conjunctive formula can be put into a normal form in which no boolean operator occurs under a sequential composition. It is then easy to extract context-free grammars from these normal forms that reverse the above translation.

Finally, we consider FLC's expressive power on finite words. We introduce the fragment LFLC which results from FLC by removing the operator $[a]$ from the syntax. Hence, LFLC is only capable of expressing the existence of certain paths. On finite words, this does not necessarily look like a restriction but it is indeed because the proof of complementation closure of FLC (on arbitrary structures) relies on the duality between $\langle a\rangle$ and $[a]$. Clearly, we have LFLC $\subseteq$ FLC on arbitrary structures, in particular on finite words.

It is also not hard to see that every context-free language is LFLC-definable. In fact, every context-free grammar gives rise to an LFLC formula using a single simultaneous least fixpoint quantifier. This inclusion is proper because of the non-closure of CSL under intersections.

Theorem 5.5 [MO99]
On finite words: CFL $\subsetneq$ LFLC.

Note that LFLC-definable properties are closed under union, intersection, and composition. Hence, LFLC is as least as expressive as the boolean closure of the context-free languages. Since the conjunction operator can occur inside of fixpoint formulas, LFLC can define languages using an unbounded number of intersections.

Theorem 5.6 [Lan02a]
On finite words: LFLC = ACFL.

We note that this result also extends to ω-words. The theory of ω-context-free languages has been introduced by Cohen and Gold [CG77]. There, a context-free grammar generates an ω-word as the limit of a derivation. Successful derivations are defined using a Muller condition on the occurrences of non-terminals in these infinite derivations. If this is extended to derivation trees in alternating context-free grammars as in [Lan02a] then it is not hard to see that these Muller conditions can be simulated by parity conditions using *latest appearance records* [DJW97].

5.3 Complexity Results

A direct consequence of Thm. 5.4 is the fact that FLC's satisfiability problem is highly undecidable. Note that this holds for the tiny fragment

HFLC already. Furthermore, the complexity results on PDL[CFG]'s model checking problem from Thm. 4.3 carry over to HFLC through the linear translations in Thm. 5.4 as well.

Theorem 5.7 [LS06]
The satisfiability problem for FLC is Σ^1_1-hard.

Theorem 5.8 [LS06]
The combined complexity of the model checking problem for HFLC is P-complete.

It was observed by Müller-Olm that the model checking problem for full FLC is decidable on finite transition systems. He also showed a lower bound of PSPACE for its data complexity [MO99, LS02]. This is not optimal though.

Theorem 5.9 [LS02, Lan06b]
The model checking problem (combined complexity) for FLC is EXPTIME-complete.

In fact, the lower bound already holds for certain fixed and non-alternating formulas (but clearly not any in HFLC).

Theorem 5.10 [Lan06b, ALS07]
The model checking problem (data complexity) for FLC is EXPTIME-complete.

The currently known best upper bound on the expression complexity is EXPTIME which is trivially inherited from the upper bound for the combined complexity. [Lan06b] states wrongly that UP$\cap$co-UP is a better upper bound, but this relies on the characterisation of FLC's model checking problem as parity games [LS02] which is not correct either.

Theorem 5.11 [Lan02a, Okh03b]
The model checking problem on finite words (combined complexity) for FLC is P-complete.

The lower bound is inherited from PDL[CFG], i.e. the word problem for context-free languages. The upper bound is obtained by extending the well-known CYK-algorithm [You67] for the word problem in context-free grammars to alternating context-free grammars.

Chapter 6

The Modal Iteration Calculus

6.1 Inflationary Fixpoint Iterations

Definition 6.1 Let $(V, \leq)$ be a complete lattice with infimum $\bot$ and joins $\sqcup$, and $f : V \to V$ not necessarily monotone. The *inflationary fixpoint* of f is defined for as $\mathtt{ifp} f := \bigsqcup_\alpha f^\alpha$, where for all ordinals α:

$$f^0 \;:=\; \bot \;, \qquad f^{\alpha+1} \;:=\; f^\alpha \sqcup f(f^\alpha) \;, \qquad f^\kappa \;:=\; \bigsqcup_{\alpha<\kappa} f^\alpha$$

with κ being a limit ordinal.

Note that the approximations trivially form an ascending chain of lattice elements.

$$f^0 \;\leq\; f^1 \;\leq\; f^2 \;\leq\; \dots$$

This chain is bound to become stationary at some ordinal level – namely at the inflationary fixpoint of f.

The Békiȅ principle states that simultaneous fixpoint operators over monotone functions have the same expressive power as parametric ones. Let $(V, \leq)$ be a lattice and consider a function $F : V \times V \to V \times V$ defined as $F(x, y) = (f_1(x, y), f_2(x, y))$. Suppose both f_1 and f_2 are monotone in each of their arguments. According to the Knaster-Tarski-Theorem [Tar55], F has a least fixpoint (x^*, y^*). The Békiȅ principle [Bék84] now reads as

$$x^* \;=\; \mu x.f_1(x, y^*) \;=\; \mu x.f_1(x, \mu y.f_2(x, y))$$

and similarly for y^*.

The proof of each direction relies on the monotonicity of f_1 and f_2. In the setting of inflationary fixpoints, monotonicity is not a prerequisite anymore. Hence, it is not surprising that such an equation does not hold for inflationary fixpoints. In general, both directions fail.

Note that the modal μ-calculus only features parametrised fixpoint operators in its ordinary form. Most of its implementations work with simultaneous ones instead, since they allow shorter representations of formulas. This is valid because of the Békič principle. A logic featuring inflationary fixpoint operators would either have to allow simultaneous ones straight away or sacrifice some of its potential in expressive power.

6.2 Syntax and Semantics

Definition 6.2 Let Σ be a finite set of action names, $\mathcal{P}$ an at most countably infinite set of atomic propositions, and $\mathcal{V}$ be a countably infinite set of variables. Formulas of the *Modal Iteration Calculus* (MIC) are given by the following grammar.

$$\varphi \ ::= \ q \mid X \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mathtt{ifp}_i(X_1, \ldots, X_n).(\varphi_1, \ldots, \varphi_n)$$

where $n \geq 1$, $1 \leq i \leq n$, $q \in \mathcal{P}$, $a \in \Sigma$ and $X, X_i \in \mathcal{V}$. Again, we use the usual abbreviations for $\mathtt{tt}$, $\mathtt{ff}$, $\varphi \wedge \psi$, $[a]\varphi$, etc. We also write $\mathtt{ifp}X.\varphi$ instead of $\mathtt{ifp}_1(X_1).(\varphi_1)$, and sometimes use a vertical rather than horizontal notation for the systems of equations between variables and formulas.

The fragment 1MIC is obtained from MIC by restricting the use of inflationary fixpoint operators to $n = 1$.

Definition 6.3 Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ be an LTS. The semantics of a MIC formula is defined inductively using environments $\rho : \mathcal{V} \to 2^{\mathcal{S}}$.

$$\begin{aligned}
[\![q]\!]^{\mathcal{T}}_{\rho} &:= \{s \in \mathcal{S} \mid q \in L(s)\} \\
[\![\varphi \vee \psi]\!]^{\mathcal{T}}_{\rho} &:= [\![\varphi]\!]^{\mathcal{T}}_{\rho} \cup [\![\psi]\!]^{\mathcal{T}}_{\rho} \\
[\![\neg\varphi]\!]^{\mathcal{T}}_{\rho} &:= \mathcal{S} \setminus [\![\varphi]\!]^{\mathcal{T}}_{\rho} \\
[\![\langle a \rangle \varphi]\!]^{\mathcal{T}}_{\rho} &:= \{s \in \mathcal{S} \mid \exists t \in \mathcal{S} \text{ s.t. } s \xrightarrow{a} t \text{ and } t \in [\![\varphi]\!]^{\mathcal{T}}_{\rho}\}
\end{aligned}$$

For the remaining case of $\mathtt{ifp}_i\vec{X}.\vec{\varphi}$ assume $\vec{X} = (X_1, \ldots, X_n)$ and $\vec{\varphi} = (\varphi_1, \ldots, \varphi_n)$. Let $f_j := \lambda T_1 \ldots \lambda T_n.[\![\varphi_j]\!]^{\mathcal{T}}_{\rho'}$ for every $j = 1, \ldots, n$ where $\rho' := \rho[X_1 \mapsto T_1, \ldots, X_n \mapsto T_n]$. Furthermore, let

$$F_{\vec{\varphi}} := \lambda T_1 \ldots \lambda T_n.\big(f_1(T_1, \ldots, T_n), \ldots, f_n(T_1, \ldots, T_n)\big)$$

Then we define

$$[\![\, \mathtt{ifp}_i \vec{X}.\vec{\varphi}\,]\!]^{\mathcal{T}}_{\rho} \;:=\; \pi_i\big(\mathtt{ifp}\ F_{\vec{\varphi}}\big)$$

where $\pi_i(T_1, \ldots, T_n) = T_i$ projects a tuple onto its i-th component.

Example 6.1 [DGK04b]

The non-regular but context-free language $\{a^n b^m \mid n \leq m\}$ on finite words of the form $a^* b^*$ is defined by the following MIC formula.

$$\begin{array}{rcl} \varphi & := & \mathtt{ifp}X.(a \wedge \psi(X)) \vee (b \wedge \Box X) \\ \psi(X) & := & \mathtt{ifp}Y.\Diamond(b \wedge \neg X) \vee \Diamond(a \wedge X \wedge Y) \end{array}$$

Here we assume that each state is labeled with either $\{a\}$ or $\{b\}$. Note that this, as well as the fact that models are linear (upto bisimulation) are expressible in $\mathcal{L}_\mu$ already.

In order to see that the above formula expresses the desired property one either computes the inflationary fixpoints by hand. Alternatively, one can consult the game-theoretic characterisation of MIC's model checking problem via *backtracking games* [DGK04a].

Example 6.2

Uniform inevitability – as mentioned in Example 5.2 – is also MIC-definable.

$$\mathtt{ifp}_1 \begin{pmatrix} X_1 & . & q \vee (\Diamond\mathtt{tt} \wedge \Box X_1 \wedge \Box\neg X_2) \\ X_2 & . & \neg X_1 \end{pmatrix}$$

expresses that q holds on all paths eventually – at the same moment.

Example 6.3

Another important non-regular property (of trees) is that of being bisimilar to a (possibly infinite) word. As said before, bisimilar models are usually considered equivalent w.r.t. the program behaviour that they model. Furthermore, model checking on words is often easier than on general trees or transition systems. Hence, replacing the tree model of a program with a word model might speed up verification tasks.

Note that a tree is bisimilar to a word iff it is balanced and within any level of the tree all nodes carry the same label. Due to our definition of (finite) words we interpret this as "for all nodes on level n there is a unique alphabet symbol a s.t. nodes on level $n+1$ are only reachable via $\xrightarrow{a}$ transitions."

It is easier to consider the complement of this. A tree is not bisimilar to a word iff it is either unbalanced or there are two nodes on the same level with outgoing transitions of different kinds.

$$\varphi_{bal} \;:=\; \mathtt{ifp}_1 \begin{pmatrix} X & . & \Box X \wedge \Box\neg Y \\ Y & . & X \end{pmatrix}$$

is satisfied by an LTS iff its unfolding to a tree is finite and balanced. We first define a formula that checks whether or not an LTS contains a finite path.

$$\varphi_{fin} \;:=\; \mathtt{ifp} X. \Box \mathtt{ff} \vee \Diamond X$$

Finally, on finite and balanced trees the following formula checks the property mentioned above: there are two nodes on the same level with two differently labeled outgoing edges.

$$\varphi_{nbis} := \bigvee_{a \neq b} \mathtt{ifp}_1 \begin{pmatrix} X_1 & . & \mathit{Reach}(\langle a \rangle \mathtt{tt} \wedge X_2 \wedge \neg X_3) \wedge \\ & & \mathit{Reach}(\langle b \rangle \mathtt{tt} \wedge X_2 \wedge \neg X_3) \\ X_2 & . & \Box X_2 \\ X_3 & . & X_2 \end{pmatrix}$$

where $\mathit{Reach}(\psi) := \mathtt{ifp} Z. \psi \vee \Diamond Z$ simply checks whether there is a path on which ψ holds eventually.

Putting this together we obtain the formula

$$\varphi_{bis} \;:=\; \varphi_{fin} \wedge \varphi_{bal} \wedge \neg \varphi_{nbis}$$

which is satisfied by an LTS iff it is bisimilar to a finite word.

6.3 Complexity Results

The situation here is comparable to that of FLC. Due to the increase in expressive power compared to the modal μ-calculus, MIC's satisfiability problem becomes highly undecidable. Its model checking problem, however, retains a comparably low complexity.

Theorem 6.1 [DGK04b]
The satisfiability problem for MIC is hard for $\bigcup_{k \in \mathbb{N}} \Sigma^0_k$ and in Σ^1_2.

The lower bound is proved by encoding natural numbers as sets of nodes in well-founded trees, and first-order arithmetic as MIC formulas. The upper bound is a consequence of the fact that MIC can be embedded into first-order logic with inflationary fixpoints. We remark that both results already hold true for 1MIC.

Theorem 6.2 [DGK04b]
The model checking problem (combined complexity) for MIC is PSPACE-complete.

Theorem 6.3 [DGK04b]
The model checking problem (expression complexity) for MIC is PSPACE-complete.

Remember that FLC's model checking problem remains hard when the formula is fixed. MIC behaves in the opposite way.

Theorem 6.4 [DGK04b]
The model checking problem (data complexity) for MIC is in P.

6.4 Expressive Power

Theorem 6.5 [DGK04b]
$\mathcal{L}_\mu \subsetneq$ 1MIC.

The inclusion holds because the inflationary fixpoint of a monotone function equals its least fixpoint. Hence, $\mathcal{L}_\mu$ can trivially be translated into 1MIC by replacing every μ with `ifp`.

Theorem 6.6 [DGK04b]
1MIC $\subsetneq$ MIC.

The inclusion is trivial. Dawar, Grädel and Kreutzer then present a MIC formula whose models are exactly the trees (upto bisimulation) of finite height and show by induction on the structure of formulas that it is not equivalent to any 1MIC formula.

It is possible to find FLC-definable properties which cannot be expressed in MIC. One example of such a property was presented in Example 5.5 describing language inclusion of nondeterministic finite automata in FLC. Dawar, Grädel and Kreutzer have shown that the *automaticity* – the least sizes of finite tree automata recognising the sets of fixed-length tree prefixes of a certain property – is at most exponential for any MIC-definable property [DGK04b]. The automaticity of the property in Example 5.5 is asymptotically worse than that.

Theorem 6.7
FLC $\not\subseteq$ MIC.

However, there is also a much simpler complexity-theoretic argument. Suppose FLC $\subseteq$ MIC, i.e. for every FLC formula φ there is a $\varphi' \in$ MIC that is equivalent to it (w.r.t. the definition of weak equivalence in FLC of course). According to Thm. 5.10 there is a $\varphi_0 \in$ FLC whose set of models is an EXPTIME-hard language. Consider φ_0' which obviously has the same set of models. According to Thm. 6.4, the set of its models is in P. But P $\neq$ EXPTIME.

Not much is known about the expressive power of MIC on finite words. For example, it is not even known whether MIC can express all context-free languages. Example 6.1 shows that it can express some, and Dawar, Grädel and Kreutzer present some 1MIC formulas expressing non-context-free properties.

Theorem 6.8 [DGK01]
On finite words: 1MIC $\not\subseteq$ CFL.

Some better bounds have been found in terms of complexity classes.

Theorem 6.9 [DGK04b]
On finite words: DTIME($O(n)$) $\subseteq$ MIC $\subseteq$ DSPACE($O(n)$).

As a consequence, every MIC-definable property is deterministic context-sensitive.

Chapter 7

Higher-Order Fixpoint Logic

7.1 Syntax and Semantics

Definition 7.1 Let $\mathcal{P} = \{p, q, \ldots\}$ be a set of atomic propositions, $\Sigma = \{a, b, \ldots\}$ be a finite set of action names, and $\mathcal{V} = \{X, Y, \ldots\}$ a set of variable names.

A $v \in \{-, +, 0\}$ is called a *variance*. The set of HFL types is the smallest set containing the atomic type Pr and being closed under function typing with variances, i.e. if σ and τ are HFL types and v is a variance, then $\sigma^v \to \tau$ is an HFL type.

Formulas of HFL are given by the following grammar:

$$\varphi \ ::= \ q \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle a\rangle\varphi \mid \varphi\ \varphi \mid \lambda(X^v : \tau).\varphi \mid \mu(X : \tau).\varphi$$

where v is a variance, τ is a type, $a \in \Sigma$, and $q \in \mathcal{P}$.

We use the standard abbreviations: $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathcal{P}$, $\mathtt{ff} := \neg\mathtt{tt}$, $\varphi \wedge \psi := \neg(\neg\varphi \wedge \neg\psi)$, $[a]\psi := \neg\langle a\rangle\neg\psi$, and $\nu X.\varphi := \neg\mu X.\neg\varphi[\neg X/X]$. We will assume that any variable without an explicit type annotation is of the ground type Pr. Also, if a variance is omitted it is implicitly assumed to be 0.

A sequence Γ of the form $X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$ where X_i are variables, τ_i are types and v_i are variances is called a *context* (we assume all X_i are distinct). An HFL formula φ has type τ in context Γ if the statement $\Gamma \vdash \varphi\colon \tau$ can be inferred using the rules of Figure 7.1. We say that φ is *well-formed* if $\Gamma \vdash \varphi\colon \tau$ for some Γ and τ. In this case τ is called the *type* of φ w.r.t. Γ. It is not hard to see that the type of a formula is unique (upto Γ), and that $\Gamma = \emptyset$ suffices for closed formulas.

$$\frac{}{\Gamma \vdash q\colon \mathrm{Pr}} \qquad \frac{v \in \{0,+\}}{\Gamma, X^v\colon \tau \vdash X\colon \tau} \qquad \frac{\Gamma^- \vdash \varphi\colon \tau}{\Gamma \vdash \neg\varphi\colon \tau} \qquad \frac{\Gamma \vdash \varphi\colon \mathrm{Pr}}{\Gamma \vdash \langle a\rangle\varphi\colon \mathrm{Pr}}$$

$$\frac{\Gamma \vdash \varphi\colon \mathrm{Pr} \quad \Gamma \vdash \psi\colon \mathrm{Pr}}{\Gamma \vdash \varphi \vee \psi\colon \mathrm{Pr}} \qquad \frac{\Gamma, X^v\colon \sigma \vdash \varphi\colon \tau}{\Gamma \vdash \lambda(X^v\colon \sigma).\varphi : (\sigma^v \to \tau)}$$

$$\frac{\Gamma \vdash \varphi : (\sigma^+ \to \tau) \quad \Gamma \vdash \psi\colon \sigma}{\Gamma \vdash (\varphi\ \psi) : \tau} \qquad \frac{\Gamma \vdash \varphi : (\sigma^- \to \tau) \quad \Gamma^- \vdash \psi\colon \sigma}{\Gamma \vdash (\varphi\ \psi) : \tau}$$

$$\frac{\Gamma \vdash \varphi : (\sigma^0 \to \tau) \quad \Gamma \vdash \psi\colon \sigma \quad \Gamma^- \vdash \psi\colon \sigma}{\Gamma \vdash (\varphi\ \psi) : \tau} \qquad \frac{\Gamma, X^+\colon \tau \vdash \varphi\colon \tau}{\Gamma \vdash \mu(X\colon \tau).\varphi : \tau}$$

Figure 7.1: Type inference rules for HFL.

Definition 7.2 For a variance v, we define its complement v^- as $+$ if $v = -$, as $-$, if $v = +$, and 0 otherwise. For a context $\Gamma = X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$, the complement Γ^- is defined as $X_1^{\bar{v_1}} : \tau_1, \ldots, X_n^{\bar{v_n}} : \tau_n$.

The semantics of a type w.r.t. a transition system $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ is a complete lattice, inductively defined on the type as

$$\begin{aligned}
[\![\, \mathrm{Pr} \,]\!]^{\mathcal{T}} &:= (2^{\mathcal{S}}, \sqsubseteq_{\mathrm{Pr}}) \qquad \text{with } \sqsubseteq_{\mathrm{Pr}} := \subseteq \\
[\![\, \sigma^v \to \tau \,]\!]^{\mathcal{T}} &:= \Big(([\![\, \sigma \,]\!]^{\mathcal{T}})^v \to_{mon} [\![\, \tau \,]\!]^{\mathcal{T}}, \sqsubseteq_{\sigma^v \to \tau} \Big) \\
&\qquad \text{with} \quad f \sqsubseteq_{\sigma^v \to \tau} g \ \text{ iff } \ \forall x \in [\![\, \sigma \,]\!]^{\mathcal{T}} : f(x) \sqsubseteq_\tau g(x)
\end{aligned}$$

where $A \to_{mon} B$ denotes the space of all *monotone* functions from A to B. A negative variance turns a lattice upside down, a positive keeps it, and a neutral variance does both at the same time.

$$(V, \sqsubseteq_\tau)^+ := (V, \sqsubseteq_\tau) \qquad (V, \sqsubseteq_\tau)^- := (V, \sqsupseteq_\tau) \qquad (V, \sqsubseteq_\tau)^0 := (V, =_\tau)$$

Definition 7.3 An environment ρ is a possibly partial map on the variable set $\mathcal{V}$. For a context $\Gamma = X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$, we say that ρ respects Γ, denoted by $\rho \models \Gamma$, if $\rho(X_i) \in [\![\, \tau_i \,]\!]^{\mathcal{T}}$ for $i \in \{1, \ldots, n\}$. We write $\rho[X \mapsto a]$ for the environment that maps X to a and otherwise agrees with ρ. If $\rho \models \Gamma$ and $a \in [\![\, \tau \,]\!]^{\mathcal{T}}$ then $\rho[X \mapsto a] \models \Gamma, X : \tau$, where X is a variable that does not appear in Γ.

For any well-typed term $\Gamma \vdash \varphi : \tau$ and environment $\rho \models \Gamma$, we define the semantics of φ inductively to be an element of $[\![\, \tau \,]\!]^{\mathcal{T}}$. In the clause for function application $(\varphi\ \psi)$ the context Γ' is Γ if $v \in \{+, 0\}$, and is Γ^- if $v = -$.

$$\begin{aligned}
[\![\,\Gamma \vdash q : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} &:= \{s \in \mathcal{S} \mid q \in L(s)\} \\
[\![\,\Gamma \vdash X : \tau\,]\!]^{\mathcal{T}}_{\rho} &:= \rho(X) \\
[\![\,\Gamma \vdash \neg\varphi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} &:= \mathcal{S} \setminus [\![\,\Gamma^- \vdash \varphi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} \\
[\![\,\Gamma \vdash \varphi \vee \psi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} &:= [\![\,\Gamma \vdash \varphi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} \cup [\![\,\Gamma \vdash \psi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} \\
[\![\,\Gamma \vdash \langle a\rangle\varphi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} &:= \{s \in \mathcal{S} \mid \\
&\qquad\qquad \exists t \in [\![\,\Gamma \vdash \varphi : \mathrm{Pr}\,]\!]^{\mathcal{T}}_{\rho} \text{ s.t. } s \xrightarrow{a} t\} \\
[\![\,\Gamma \vdash \lambda(X^v : \tau).\varphi : \tau^v \to \tau'\,]\!]^{\mathcal{T}}_{\rho} &:= \lambda x \in [\![\,\tau\,]\!]^{\mathcal{T}}.[\![\,\Gamma, X^v : \tau \vdash \varphi : \tau'\,]\!]^{\mathcal{T}}_{\rho[X \mapsto d]} \\
[\![\,\Gamma \vdash \varphi\ \psi : \tau'\,]\!]^{\mathcal{T}}_{\rho} &:= \big([\![\,\Gamma \vdash \varphi : \tau^v \to \tau'\,]\!]^{\mathcal{T}}_{\rho}\big)\ \big([\![\,\Gamma' \vdash \psi : \tau\,]\!]^{\mathcal{T}}_{\rho}\big) \\
[\![\,\Gamma \vdash \mu(X : \tau)\varphi : \tau\,]\!]^{\mathcal{T}}_{\rho} &:= \bigsqcap \{x \in [\![\,\tau\,]\!]^{\mathcal{T}} \mid \\
&\qquad\qquad [\![\,\Gamma, X^+ : \tau \vdash \varphi : \tau\,]\!]^{\mathcal{T}}_{\rho[X \mapsto x]} \sqsubseteq_{\tau} x\}
\end{aligned}$$

Definition 7.4 We consider fragments of formulas that can be built using restricted type ranks and maximal arities of types only. Note that by right-associativity of the function type operator $\to$, every HFL type is isomorphic to one of the form $\tau_1 \to \ldots \to \tau_m \to \mathrm{Pr}$. Let

$$\begin{aligned}
\mathit{rank}(\tau_1 \to \ldots \to \tau_m \to \mathrm{Pr}) &:= \max\{\mathit{rank}(\tau_i) + 1 \mid i = 1, \ldots, m\} \\
\mathit{mar}(\tau_1 \to \ldots \to \tau_m \to \mathrm{Pr}) &:= \max\{m, \max\{\mathit{mar}(\tau_i) \mid i = 1, \ldots, m\}\}
\end{aligned}$$

where $m \geq 0$ and $\max \emptyset = 0$. Now let

$$\begin{aligned}
\mathrm{HFL}^{k,m} := \{\ \varphi \in \mathrm{HFL} \mid\ \vdash \varphi : \mathrm{Pr}\ &\text{using types } \tau \text{ with } \mathit{rank}(\tau) \leq k, \\
&\mathit{mar}(\tau) \leq m \text{ only }\}
\end{aligned}$$

Note that $\mathrm{HFL} = \mathrm{HFL}^{0,0} \cup \bigcup_{k\geq 1} \bigcup_{m \geq 1} \mathrm{HFL}^{k,m}$ since higher-order types necessarily need to have arguments and vice-versa. We also write HFL^0 instead of $\mathrm{HFL}^{0,0}$ and HFL^k for $\bigcup_{m\geq 1} \mathrm{HFL}^{k,m}$ if $k \geq 1$.

Example 7.1 [VV04]

HFL is capable of expressing *assume-guarantee properties*. Let $\varphi(X)$ and $\psi(Y)$ be two formulas. Define

$$s \models \nu X.\varphi \triangleright \nu Y.\psi \quad \text{iff} \quad \forall k \in \mathbb{N} : s \models \varphi^k(\mathtt{tt}) \Rightarrow s \models \psi^{k+1}(\mathtt{tt})$$

where $\varphi^0(\mathtt{tt}) := \mathtt{tt}$ and $\varphi^{k+1}(\mathtt{tt}) := \varphi(\varphi^k(\mathtt{tt}))$, etc.

Viswanathan and Viswanathan note that the modal μ-calculus is not closed under assume-guarantee properties. For example, let $\varphi(X) := \langle a\rangle X$ and $\psi(Y) := \langle b\rangle Y$, then $\nu X.\langle a\rangle X \triangleright \nu Y.\langle b\rangle Y$ expresses "if there is an a-path of length k for some k then there is also a b-path of length $k+1$". This is not a regular property.

In general, we have

$$\nu X.\varphi(X) \triangleright \nu Y.\psi(Y) \;\equiv\; \big(\nu Z.\lambda X.\lambda Y.(\neg X \vee Y) \wedge Z\ \varphi(X)\ \psi(Y)\big)\ \mathtt{tt}\ \psi(\mathtt{tt})$$

with appropriate type annotations. Hence, assume-guarantee properties are HFL^1-definable

Example 7.2 [LS05]
Let $2^n_0 := n$ and $2^n_{m+1} := 2^{2^n_m}$. For any $m \in \mathbb{N}$, there is an HFL formula φ_m expressing the fact that there is a maximal path of length 2^1_m (number of states on this path) through a transition system. It can be constructed using a typed version of the Church numeral 2. Let $\tau_0 = \mathrm{Pr}$ and $\tau_{i+1} = \tau_i \to \tau_i$. For $i \geq 1$ define ψ_i of type τ_{i+1} as $\lambda(F : \tau_i).\lambda(X : \tau_{i-1}).F\ (F\ X)$. Then

$$\varphi_m := \psi_m\ \psi_{m-1}\ \ldots\ \psi_1\ \big(\lambda X.\Diamond X\big)\ \Box\mathtt{ff}$$

expresses the desired property. Note that for any $m \in \mathbb{N}$, φ_m is of size linear in m.

Example 7.3
Recall Ex. 6.3 which shows that MIC is capable of expressing the property of being bisimilar to a word model. This can also be done in HFL using function arguments as stacks rather than inflationary fixpoints.

Remember that a tree is not bisimilar to a word iff there are two nodes on the same level with outgoing transitions of different kinds.

$$\varphi \;:=\; \bigvee_{a,b\in\Sigma, a\neq b} \big(\mu X.\lambda A.\lambda B.(A \wedge B) \vee (X\ \Diamond A\ \Diamond B)\big)\ \langle a\rangle\mathtt{tt}\ \langle b\rangle\mathtt{tt}$$

This formula is equivalent by fixpoint approximation and β-reduction to

$$\bigvee_{a,b\in\Sigma, a\neq b} (\langle a\rangle\mathtt{tt} \wedge \langle b\rangle\mathtt{tt}) \vee (\Diamond\langle a\rangle\mathtt{tt} \wedge \Diamond\langle b\rangle\mathtt{tt}) \vee (\Diamond\Diamond\langle a\rangle\mathtt{tt} \wedge \Diamond\Diamond\langle b\rangle\mathtt{tt}) \vee \ldots$$

Since HFL is closed under negation it is also possible to express bisimilarity to a word.

7.2 Expressive Power and Complexity Results

Theorem 7.1
$\mathcal{L}_\mu = \text{HFL}^0$.

Note that $\mathcal{L}_\mu$ is a syntactical fragment of HFL and that every subformula of a $\mathcal{L}_\mu$ formula has type rank 0 in HFL. Furthermore, any HFL^0 formula cannot contain a subformula of type rank ≥ 1. Hence, it cannot contain a subformula of the form $\lambda(X : \tau).\varphi$ or $\varphi\ \psi$. It is easy to see that the remaining syntax only allows to build formulas that are in $\mathcal{L}_\mu$ already.

Theorem 7.2 [VV04]
$\text{FLC} \subseteq \text{HFL}^{1,1}$.

This is not surprising. Note that the semantics of any FLC formula is a function of type $\text{Pr} \to \text{Pr}$, i.e. of type rank 1 and maximal arity 1. The translation from FLC to $\text{HFL}^{1,1}$ is straight-forward.

Viswanathan and Viswanathan also show that there are HFL-definable properties which are not FLC-definable using a diagonalisation argument. Such a formula is of type rank 2, though. We believe that FLC is already strictly contained in HFL^1. The formula presented in Example 7.3 does not seem to be expressible in FLC which remains to be proved. However, it can easily be expressed in FLC with inverse modalities. Hence, such a proof would already separate FLC from FLC with inverse modalities on trees.

Thm. 7.2 raises the question after the relationship between MIC and HFL. Unfortunately, we can only answer this partially.

Theorem 7.3
$\text{MIC} \subsetneq \text{HFL}^1$ on finite structures.

Strictness is a simple consequence of Thm. 6.7 – the fact that there are FLC-definable properties which are not expressible in MIC. The type rank 1 is the price for the transition from a non-monotone to a monotone function. The width of an inflationary fixpoint operator – the number of simultaneously defined variables – translates into a number of arguments to a function of type rank 1. Thus, there is probably no fixed m s.t. we can embed MIC into $\text{HFL}^{k,m}$ as it is the case for FLC. For a detailed proof of this inclusion see Appendix A. An immediate consequence is the fact that 1MIC can be embedded using fixed arities only.

Theorem 7.4
1MIC $\subsetneq$ HFL^{1,1} on finite structures.

The satisfiability problem for logics like PDL[CFG] and MIC, that are rather weak in comparison to HFL, is already beyond the arithmetic hierarchy. Hence, HFL is highly undecidable as well. Since HFL incorporates functions of arbitrary rank it seems possible that its satisfiability problem (or that of HFL^k for some fixed and low k already) is even beyond the analytical hierarchy. However, Grädel, Dawar, and Kreutzer's encoding of the arithmetic hierarchy in MIC does not seem to extend easily to second-order quantification since $2^{\mathbb{N}}$ is not enumerable.

Example 7.2 provides a hint at the fact that HFL's model checking problem is difficult. It is non-elementary in general, but still fairly low in the Grzegorczyk hierarchy [Grz53], namely included in $\mathcal{E}_4$.

Theorem 7.5 [LS05, ALS07]
The model checking problem for HFL (combined complexity) is in DTIME$(2^{O(n)}_{O(n)})$.

Viswanathan and Viswanathan remarked that HFL's model checking problem is decidable on finite transition systems $\mathcal{T}$. Note that every lattice $[\![\, \tau \,]\!]^{\mathcal{T}}$ is finite, but possibly of size $2_k^{O(n)}$ where n is the number of states of $\mathcal{T}$ and $k = \mathit{rank}(\tau)$. Thus, model checking can be performed by fixpoint iteration in these finite lattices.

A more careful analysis using fixpoint elimination yields a reduction from HFL's model checking problem to the problem of solving a (rather large) reachability game.

Theorem 7.6 [ALS07]
For all $k \geq 1, m \geq 2$, the model checking problem for HFL^{k,m} (combined complexity) is kEXPTIME-complete.

The reduction in the lower bound from the word problem for alternating space bounded Turing Machines can be refined to yield HFL formulas which do not depend on the machine's input word anymore. The price to pay is a slightly higher arity of the resulting types.

Theorem 7.7 [ALS07]
For all $k \geq 1, m \geq 4$, the model checking problem for HFL^{k,m} (data complexity) is kEXPTIME-complete.

The size of the resulting reachability game in the general upper bound, for a transition system of size n and an HFL^{k,m} formula φ is

$$O(n^2)|\varphi|^2(m^{(k-1)m^{k-2}} 2_k^{n(k-1+m)^{k-1}})^{2(m+|\varphi|)}$$

Note how this collapses when, in addition to k and m, the parameter n is also fixed. Hence, it is the data complexity that makes the problem difficult rather than its expression complexity.

Theorem 7.8 [ALS07]
For all $k, m \geq 1$, the model checking problem for HFL^{k,m} (expression complexity) is in EXPTIME.

The fact that the data complexity of HFL is non-elementary yields another hierarchy result about expressive power.

Theorem 7.9 [ALS07]
For all $k \in \mathbb{N}$: HFL$^k \subsetneq$ HFL^{k+1}.

Formulas that witness the strictness of this inclusion are constructed in the proof of the lower bound in Thm. 7.8. The set of their models is kEXPTIME-hard. If they were equivalent to an HFL^{k-1} formula then this would also be an HFL$^{k-1,m}$ formula for some m. But their sets of models are recognisable in $(k-1)$EXPTIME which is strictly less than kEXPTIME according to the time-hierarchy theorem.

Chapter 8

Conclusions

8.1 Summary

This thesis provides an overview of some temporal logics that are capable of expressing non-regular properties. It focuses on extensions of the well-known modal μ-calculus and fragments thereof, in particular

- *Propositional Dynamic Logic of Context-Free Programs* which extends ordinary Propositional Dynamic Logic with the programs formed by context-free grammars;
- *Fixpoint Logic with Chop* which extends the modal μ-calculus with an operator for sequential composition;
- the *Modal Iteration Calculus* which extends the modal μ-calculus by replacing least with inflationary fixpoint quantifiers; and
- *Higher-Order Fixpoint Logic* which merges the modal μ-calculus with a *simply typed λ-calculus*.

Regarding relative expressive power, it turns out that the latter subsumes all others, the second subsumes the first and the second and third are probably incomparable. From this point of view, it is not surprising that the complexity of model checking formulas of such logics is low in the first case, very high in the last case and mildly non-tractable in the two others.

Both aspects are summarised in more detail in the following two sections where we put these logics, their fragments and the related formalisms mentioned earlier into two pictures showing their relationships over arbitrary structures, resp. finite words. We also compare them w.r.t. their model checking problems regarding both combined and data complexities. The latter carries a clear message: model checking a non-regular property need not be vastly more expensive than model checking a regular property – it simply depends on which logic is chosen. Note that both PDL[CFG] and MIC are capable of expressing non-regular properties, and both their data complexities are in P.

8.2 The Family of Temporal Logics Beyond Regularity

Fig. 8.1 shows the relationships between the branching time temporal logics of non-regular properties that are featured in the previous chapters, and how they link to the modal μ-calculus.

Strict inclusions are indicated using dashed lines. A continuous line upwards shows an inclusion which is not known to be strict. The following table contains properties that witness the non-inclusion relations in this picture.

non-inclusion			example property
PDL[CFG]	$\not\subseteq$	$\mathcal{L}_\mu$	"there is a path of the form $a^n b^n$"
$\mathcal{L}_\mu$	$\not\subseteq$	PDL[CFG]	"on all paths eventually ..."
FLC^0	$\not\subseteq$	PDL[CFG]	"on all paths eventually ..."
FLC^{k+1}	$\not\subseteq$	FLC^k	description of winning strategies in FLC model checking games with k priorities
1MIC	$\not\subseteq$	$\mathcal{L}_\mu$	"there is a path of the form $a^n b^m$" for $m \geq n$
MIC	$\not\subseteq$	1MIC	bisimilarity of subtrees
FLC	$\not\subseteq$	MIC	language inclusion between NFA
$\text{HFL}^{1,1}$	$\not\subseteq$	MIC	language inclusion between NFA
HFL^{k+1}	$\not\subseteq$	HFL^k	description of alternating space bounded Turing Machines

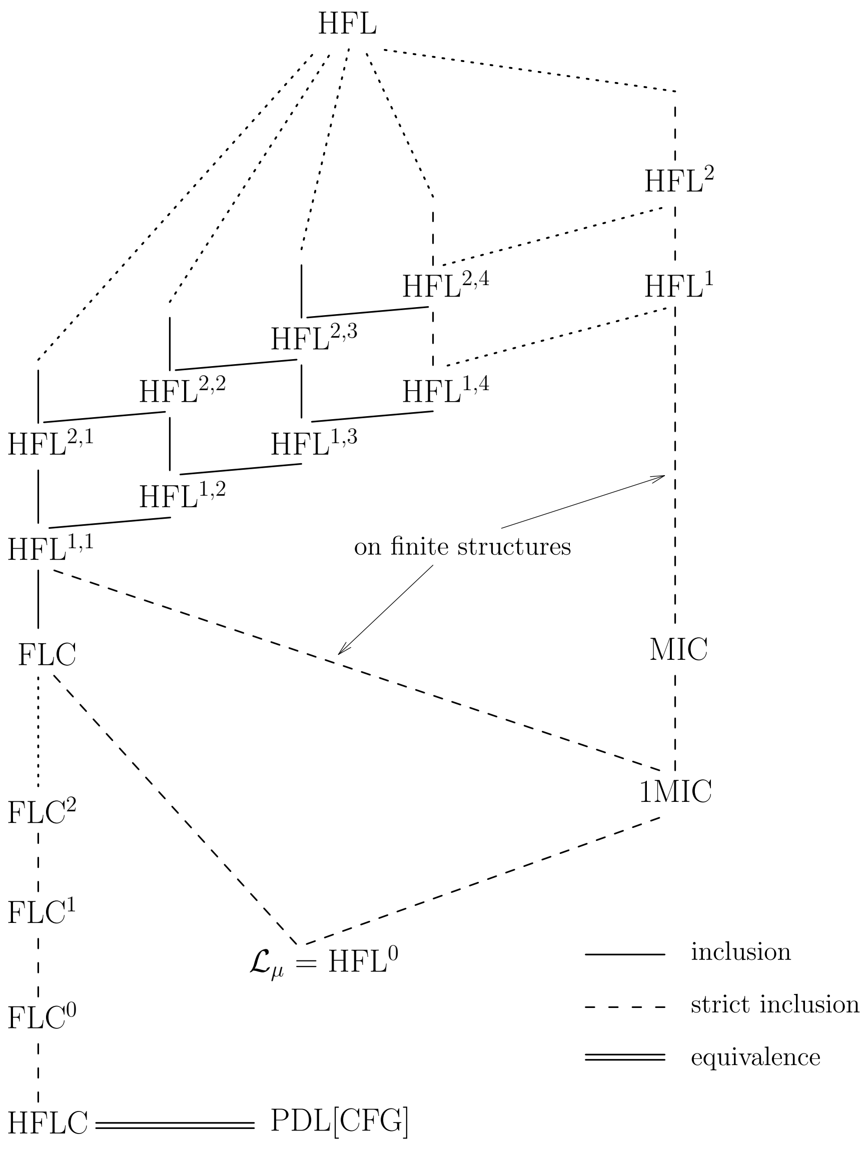


Figure 8.1: Branching Time Temporal Logics of Non-Regular Properties.

Fig. 8.2 contains the corresponding picture for linear time temporal logics, complexity classes and algebraic characterisations of languages of finite words.

8.3 The Complexity of Model Checking a Non-Regular Property

The table in Fig. 8.3 summarises the results on the model checking complexities for the branching time temporal logics PDL[CFG], FLC, MIC, HFL and their fragments mentioned in the previous chapters. It also indicates whether the known upper bounds could be matched by lower bounds yet.

Here we consider not only the combined complexity but also the data complexity since it is a more realistic measure for verification problems.

8.4 Open Questions

Fig. 8.1 and Fig. 8.2 indicate that there still are some open problems regarding the expressive power of temporal logics beyond regularity.

(1) Is FLC strictly included in HFL^{1,1}?

We suspect the answer is yes. In fact, it is very easy to see that HFL^{1,1} can do more on the level of functions. The semantics of any FLC formula is only a *monotone* function of type Pr $\to$ Pr. HFL^{1,1} can build formulas using non-monotone functions like $\lambda X.\neg X$ though. However, this does not answer the question at hand because definability in FLC and HFL^{1,1} is not directly concerned with such functions: is there an HFL^{1,1} formula of type Pr that is not equivalent to any FLC formula? I.e. is it possible to use non-monotone functions to define a sequence of elements of a family of lattices that cannot be defined using a fixed combination of monotone functions only?

(2) Are there MIC-definable properties that are not FLC-definable?

Note that the opposite holds. Hence, a positive answer would entirely separate MIC and FLC and show that inflationary fixpoints and sequential composition are – at least when it comes to modal logics – two entirely different mechanisms.

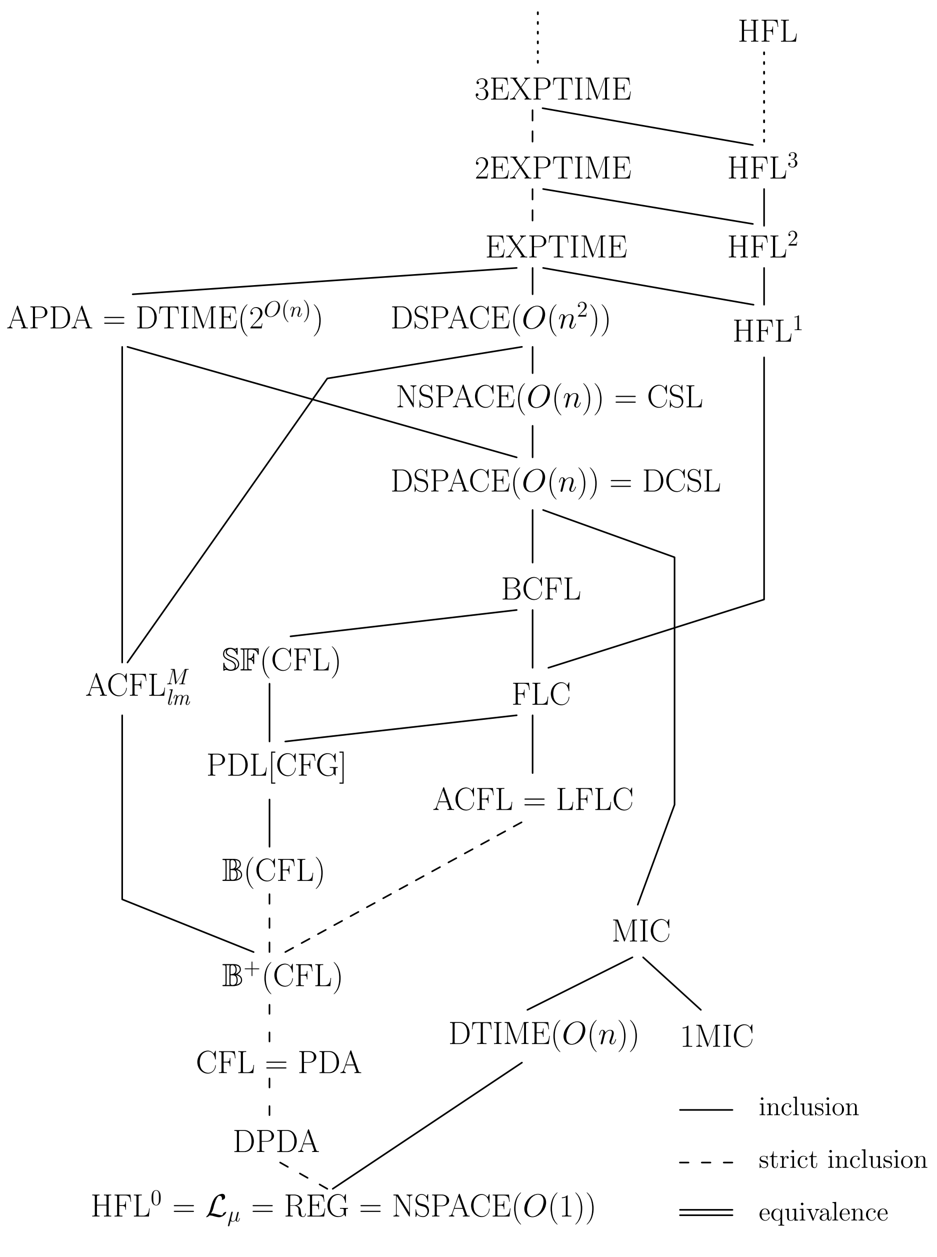


Figure 8.2: Non-Regular Properties of Finite Words.

logic	combined complexity	optimal	data complexity	optimal
PDL[CFG]	P	yes	P	
FLC^k	EXPTIME	yes	EXPTIME	yes
FLC	EXPTIME	yes	EXPTIME	yes
1MIC	PSPACE	yes	P	
MIC	PSPACE	yes	P	
HFL^0	UP$\cap$co-UP		P	yes
HFL^{k,m}	kEXPTIME	for $m \geq 2$	kEXPTIME	for $m \geq 4$
HFL	non-elemen.	yes	non-elemen.	yes

Figure 8.3: Combined and data complexities of temporal logics beyond regularity.

> *(3) Is MIC included in HFL^1 over arbitrary transition systems?*

Equally, can inflationary fixpoints in a complete lattice V always be defined using least or greatest fixpoints in the lattice of functions $V \to V$ with pointwise order?

> *(4) Are there fixpoint alternation hierarchies in HFL, HFL^k for $k \geq 1$?*

This might not have much practical relevance since type ranks and, to some extend, arities are the cause for HFL's high complexity.

> *(5) How do $ACFL^M_{lm}$, $\mathbb{SF}(CFL)$, ACFL and MIC relate to each other?*

This would reduce the width of the picture in Fig. 8.2. It would also provide some more insight into the interaction between universal quantification and sequential composition in context-free grammars.

(6) Which inclusions are strict between $\mathbb{B}^+(CFL)$ *and* EXPTIME*?*

This of course is not an easy question. The strictness of the complexity-theoretic inclusions in there have been open for quite a while without much hope for a settling answer. Regarding the inclusions between various grammar formalisms – including question (5) above – we also refer to Okhotin's list of open problems in boolean and conjunctive grammars [Okh06].

(7) Is the type rank hierarchy in HFL strict over finite words as well?

This seems quite likely given the situation depicted in Fig. 8.2. The strict exponential-time hierarchy implies that for all $k \geq 1$ we have $\text{HFL}^k \subsetneq k\text{EXPTIME}$ or $\text{HFL}^k \subsetneq \text{HFL}^{k+1}$ on finite words. The strictness of the second inclusion would follow immediately if the reduction from the word problem for alternating Turing machines to HFL's model checking problem could be modified to use the Turing Machine's input word as the transition system.

Related to that is the following question.

(8) Are there arity hierarchies in HFL, resp. each HFL^k*?*

For higher-order predicate logics, several hierarchy results regarding the arity of higher-order relations have been found [Gro96, FT06]. These are, for example, obtained by diagonalisation similar to the technique used in Thm. 3.9.

(9) Is the alternation hierarchy in FLC strict over finite models and over linear words?

Unfortunately, the proof of the strictness of the alternation hierarchy only works for infinite models [Lan06a]. It does not directly transfer to finite models as in the case of the modal μ-calculus due to the lack of the finite model property.

Similarly, on linear models the alternation hierarchy in the modal μ-calculus collapses down to level 0. This relies on the fact that on ω-words parity automata are equi-expressive to weak Büchi automata [KV98] which can be translated into alternation-free μ-calculus formulas. Remember that it is even unknown whether alternating context-free grammars define the same languages as alternating push-down automata. This, however, might be necessary in order to remove alternation from FLC formulas using translations to and from automata.

8.5 Further Work

The unanswered questions of the previous section clearly provide material for ongoing research on temporal logics for non-regular properties. These are mainly concerned with their theoretical aspects though. There are, however, also practical considerations.

The table in Fig. 8.3 shows that model checking a fixed non-regular property need not be intractable. It simply depends on the property and the logic that it can be formalised in. Clearly, this statement should be evaluated empirically in the form of a verification tool for non-regular properties. It is not hard to extend symbolic model checking algorithms for the modal μ-calculus to PDL[CFG] and MIC. For FLC and HFL^1 this is slightly more tricky because a naïve extension leads to a guaranteed exponential (in the size of the underlying transition system) running time. A less naïve approach employs a technique called *neediness analysis* that is also found in abstract program interpretation. This works for HFL^1 and, hence, covers other specification formalisms like FLC as well.

Another direction to take for further work concerns the pragmatics of the presented logics. The most successful logics in verification – in terms of usage – are those that provide the most intuitive formulas, e.g. LTL and CTL, and without a doubt these two aspects are correlated. From this point of view, the logics presented here stand very little chances of being used successfully for the verification of non-regular properties. PDL[CFG] may be the only exception here. However, it is also the least expressive of them.

However, these logics need not be used at a front-end of a verification tool. Just like LTL and CTL can be seen as readable fragments of the less readable modal μ-calculus, fragments of FLC, MIC and HFL^1 that can be given a better syntax without explicit fixpoint operators ought to be found. An example is given by FairCTL, the extension of CTL with fairness constraints over path quantifiers [EL87]. It is well-known that CTL cannot express fairness, i.e. the fact on all execution paths some (atomic) statement holds infinitely often. This can be done in LTL, but model checking LTL is PSPACE-hard whereas it is in P for CTL. FairCTL provides a way out of this dilemma. In this logic one can form statements like $\mathtt{A}_\varphi\mathtt{G}\psi$ which read as "all paths on which φ holds infinitely often satisfy ψ everywhere".

One possibility for the specification of non-regular properties in a logic with a friendly syntax is to index CTL path quantifiers with non-regular linear time properties. There, a formula of the form $\mathtt{A}_\varphi\mathtt{G}\psi$ would read as "all paths that satisfy φ also satisfy ψ everywhere" where φ is a non-regular property of ω-words. It remains to be seen which logics are expressive enough to subsume such an extension of CTL as a fragment, and whether this could lead to reasonably efficient model checking procedures.

An application of such a logic may be found in verification of infinite-state-systems through abstraction. There, the abstraction process – the transformation of an infinite-state-system to a finite-state one – often destroys too much of the systems structure such that model checking the original property on the finite-state structure provides no useful information about the satisfaction of the property on the infinite-state structure. In order to restrict this loss of information it has been suggested to incorporate fairness constraints into the property such that only fair paths of the finite-state system are considered in order to obtain information about the infinite-state one [BIS04]. Another appropriate choice to exclude traces that do not occur in the concrete system may be to incorporate a logical description of the infinite-state system into the formula. This would clearly have to be a non-regular property in general.

Index

λ-calculus
    simply typed, 10, 57

abstraction, 6
alternation, 63
    depth, 40
analytic hierarchy, 20
arithmetic
    first-order, 19
    second-order, 19
arithmetic hierarchy, 20
automaticity, 48
automaton
    alternating push-down, 28, 63
    büchi, 34
    deterministic push-down, 15
    finite, 14
    non-deterministic finite, 39
    push-down, 15
    rabin, 6, 17

backtracking games, 46
bisimilar, 13
bisimulation, 13
    -invariance, 6, 13

Chomsky hierarchy, 15
chomsky hierarchy, 15
closure
    boolean, 23
    positive boolean, 23
    star-free, 23
complexity
    combined, 17
    data, 9, 18
    expression, 18
complexity class, 14
context, 50

derivation, 25, 26

equivalence, 14
    strong, 37
    weak, 37

finite model property, 21
fixpoint
    inflationary, 44
fixpoint logic with chop, 9, 36, 57
    heterogeneous, 41
formula, 14
    conjunctive, 40
    disjunctive, 40
function
    monotone, 51

grammar, 14
    alternating, 24
    alternating context-free, 25, 26, 63
    auto-presentable, 29
    boolean, 27
    conjunctive, 26
    context-free, 24
    linear erasing, 29

higher-order fixpoint logic, 10, 57

kripke structure, 4, 11

labeled transition system, 11
language, 14
  context-free, 15
  context-sensitive, 15
  deterministic context-sensitive, 15
  formal, 14
  recursively enumerable, 15
  regular, 15
latest appearance record, 42
logic, 14

modal μ-calculus, 5, 6, 10, 15
modal iteration calculus, 9, 45, 57
model checking, 5, 17
  bounded, 6
monadic second-order logic, 6, 17

predicate transformer, 10, 36
property, 14
  assume-guarantee, 52
  regular, 6, 14, 17
propositional dynamic logic, 4, 33
  of context-free programs, 9, 32, 57
  of non-regular programs, 9

starting state, 12

temporal logic, 4
  branching time, 4
  linear time, 4
tense logic, 4
turing machine, 14
type, 50

uniform inevitability, 38, 46

variance, 50

well-formed, 50
word
  finite, 12

Bibliography

[ALS07] R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3:1–33, 2007.

[And94] H. R. Andersen. Model checking and Boolean graphs. *TCS*, 126(1):3–30, 1994.

[Arn99] A. Arnold. The modal μ-calculus alternation hierarchy is strict on binary trees. *RAIRO - Theoretical Informatics and Applications*, 33:329–339, 1999.

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20(3):207–226, 1983.

[BB89] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Proc. Coll. on Temporal Logic in Specification*, volume 398 of *LNCS*, pages 62–73. Springer, 1989.

[Bék84] H. Békiĉ. *Programming Languages and Their Definition, Selected Papers*, volume 177 of *LNCS*. Springer, 1984.

[BIS04] D. Bosnacki, N. Ioustinova, and N. Sidorova. Using fairness to make abstractions work. In *Proc. 11th Int. Workshop on Model Checking Software, SPIN'04*, volume 2989 of *LNCS*, pages 198–215. Springer, 2004.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Conf. Record of the 13th Annual ACM Symp. on Principles of Programming Languages, POPL'86*, pages 173–183. ACM, ACM, 1986.

[CBRZ01] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages, POPL'77*, pages 238–252, 1977.

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. In *Proc. 10th Symp. on Principles of Programming Languages, POPL'83*, pages 117–126. ACM, 1983.

[CG77] R. S. Cohen and A. Y. Gold. Theory of ω-languages. *Journal of Computer and System Sciences*, 15:169–184, 185–208, 1977.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

[CH93] S. Christensen and H. Hüttel. Decidability issues for infinite-state processes – a survey. *Bulletin of the EATCS*, 51:155–167, 1993.

[Cho62] N. Chomsky. Formal properties of grammars. In R. R. Bush, E. H. Galanter, and R. D. Bruce, editors, *Handbook of Mathematic Psychology*, volume 2, pages 323–418. John Wiley & Co., 1962.

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Cle90] R. Cleaveland. Tableau-based model checking in the propositional μ-calculus. *Acta Informatica*, 27(8):725–748, 1990.

[CS92] R. Cleaveland and B. Steffen. A linear–time model–checking algorithm for the alternation–free modal μ–calculus. In *Proc. 3rd Int. Conf. on Computer Aided Verification, CAV'91*, volume 575 of *LNCS*, pages 48–58. Springer, 1992.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *TCS*, 126(1):77–96, 1994.

[DGG97] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[DGK01] A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. In L. Fribourg, editor, *Proc. 15th Workshop on Computer Science Logic, CSL'01*, LNCS, pages 277–291, Paris, France, 2001. Springer.

[DGK04a] A. Dawar, E. Grädel, and S. Kreutzer. Backtracking games and inflationary fixed points. In *Proc. Int. Coll. on Automata, Languages and Programming, ICALP'04*, volume 3142 of *LNCS*, pages 420–432. Springer, 2004.

[DGK04b] A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logics. *ACM Transactions on Computational Logic*, 5(2):282–315, 2004.

[DJW97] S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How much memory is needed to win infinite games? In *Proc. 12th Symp. on Logic in Computer Science, LICS'97*, pages 99–110. IEEE, 1997.

[EC82] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[EH85] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1991. IEEE.

[EJ00] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 2000.

[EL87] E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.

[Eme87] E. A. Emerson. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 24(2):77–79, 1987.

[FL79] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[FT06] F. A. Ferrarotti and J. M. Turull Torres. Arity and alternation: A proper hierarchy in higher order logics. In *Proc. 4th Int. Symp. on Foundations of Information and Knowledge Systems, FoIKS'06*, volume 3861 of *LNCS*, pages 92–115. Springer, 2006.

[GR63] S. Ginsburg and G. F. Rose. Some recursively unsolvable problems in ALGOL-like languages. *Journal of the ACM*, 10:175–195, 1963.

[Gro96] M. Grohe. Arity hierarchies. *Annals of Pure and Applied Logic*, 82(2):103–163, 1996.

[Grz53] A. Grzegorczyk. *Some classes of recursive functions.* Rozprawy Mate. IV. Warsaw, 1953.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[HPS83] D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[HU80] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, N. Reading, MA, 1980.

[IJW92] O. H. Ibarra, T. Jiang, and H. Wang. A characterization of exponential-time languages by alternating context-free grammars. *TCS*, 99(2):301–315, 1992.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[Jur98] M. Jurdziński. Deciding the winner in parity games is in $UP \cap$co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In *Proc. 7th Conf. on Concurrency Theory, CONCUR'96*, volume 1119 of *LNCS*, pages 263–277. Springer, 1996.

[Kam68] H. W. Kamp. *On tense logic and the theory of order.* PhD thesis, Univ. of California, 1968.

[KNR06] V. Kountouriotis, Ch. Nomikos, and P. Rondogiannis. Well-founded semantics for boolean grammars. In *Proc. 10th Int. Conf. on Developments in Language Theory, DLT'06*, volume 4036 of *LNCS*, pages 203–214. Springer, 2006.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983.

[Kre04] S. Kreutzer. Expressive equivalence of least and inflationary fixed-point logic. *Ann. Pure Appl. Logic*, 130(1-3):61–78, 2004.

[Kri63] S. Kripke. Semantical considerations on modal logic. *Acta philosphica fennica*, 16:83–94, 1963.

[KV98] O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th Annual ACM Symp. on Theory of Computing, STOC'98*, pages 224–233. ACM Press, 1998.

[KVW00] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[Lan02a] M. Lange. Alternating context-free languages and linear time μ-calculus with sequential composition. In *Proc. 9th Workshop on Expressiveness in Concurrency, EXPRESS'02*, volume 68.2 of *ENTCS*, pages 71–87. Elsevier, 2002.

[Lan02b] M. Lange. Local model checking games for fixed point logic with chop. In *Proc. 13th Conf. on Concurrency Theory, CONCUR'02*, volume 2421 of *LNCS*, pages 240–254. Springer, 2002.

[Lan05] M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

[Lan06a] M. Lange. The alternation hierarchy in fixpoint logic with chop is strict too. *Information and Computation*, 204(9):1346–1367, 2006.

[Lan06b] M. Lange. Three notes on the complexity of model checking fixpoint logic with chop. *R.A.I.R.O. – Theoretical Informatics and Applications*, ??(??):??–??, 2006. (to appear).

[LLS84] R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, 1984.

[LS02] M. Lange and C. Stirling. Model checking fixed point logic with chop. In M. Nielsen and U. H. Engberg, editors, *Proc. 5th Conf. on Foundations of Software Science and Computation Structures, FOSSACS'02*, volume 2303 of *LNCS*, pages 250–263, Grenoble, France, 2002. Springer.

[LS05] M. Lange and R. Somla. The complexity of model checking higher order fixpoint logic. In *Proc. 30th Int. Symp. on Math. Foundations of Computer Science, MFCS'05*, volume 3618 of *LNCS*, pages 640–651. Springer, 2005.

[LS06] M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Information Processing Letters*, 100(2):72–75, 2006.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[MHHO05] E. Moriya, D. Hofbauer, M. Huber, and F. Otto. On state-alternating context-free grammars. *TCS*, 337(1–3):183–216, 2005.

[Mil89] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.

[MO99] M. Müller-Olm. A modal fixpoint logic with chop. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99*, volume 1563 of *LNCS*, pages 510–520. Springer, 1999.

[Mor89] E. Moriya. A grammatical characterization of alternating pushdown automata. *TCS*, 67(1):75–85, 1989.

[Okh01] A. Okhotin. Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4):519–535, 2001.

[Okh03a] A. Okhotin. Boolean grammars. In *Proc. 7th Int. Conf. on Developments in Language Theory, DLT'03*, volume 2710 of *LNCS*, pages 398–410. Springer, 2003.

[Okh03b] A. Okhotin. A recognition and parsing algorithm for arbitrary conjunctive grammars. *TCS*, 1–3(302):365–399, 2003.

[Okh04] A. Okhotin. Boolean grammars. *Information and Computation*, 194(1):19–48, 2004.

[Okh06] A. Okhotin. Nine open problems on conjunctive and boolean grammars, 2006. `http://www.cs.queensu.ca/~okhotin/boolean/boolean_nine_problems.pdf`.

[Pra78] V. R. Pratt. A practical decision method for propositional dynamic logic. In *Proc. 10th Symp. on Theory of Computing, STOC'78*, pages 326–337, 1978.

[Pri57] A. N. Prior. *Time and modality.* Oxford University Press, Oxford, UK, 1957.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th Symp. on Programming*, volume 137 of *LNCS*, pages 337–371. Springer, 1982.

[Rab69] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. of Amer. Math. Soc.*, 141:1–35, 1969.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, 1967.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.

[SS98] P. Stevens and C. Stirling. Practical model-checking using games. In B. Steffen, editor, *Proc. 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tho75] S. Thomason. Reduction of second-order logic to modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:107–114, 1975.

[Var88] M. Y. Vardi. A temporal fixpoint calculus. In ACM, editor, *Proc. Conf. on Principles of Programming Languages, POPL'88*, pages 250–259, NY, USA, 1988. ACM Press.

[Var96] M. Y. Vardi. *An Automata-Theoretic Approach to Linear Temporal Logic*, volume 1043 of *LNCS*, pages 238–266. Springer, New York, NY, USA, 1996.

[VS85] M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing, STOC'85*, pages 240–251, Baltimore, USA, 1985. ACM.

[VV04] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004.

[Wot73] D. Wotschke. The boolean closures of deterministic and nondeterministic contextfree languages. In *3. Jahrestagung der Gesellschaft für Informatik e.V.*, volume 1 of *LNCS*, pages 113–121. Springer, 1973.

[You67] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):372–375, 1967.

Appendix A

Unpublished Proofs

A.1 From the Modal Iteration Calculus to Higher-Order Fixpoint Logic

Before we present an inductive translation we simplify matters by extracting the most difficult case – that of inflationary fixpoint operators of course – into a general lattice-theoretical lemma.

Inflationary Fixpoints and Higher-Order Least Fixpoints
Suppose $(V, \leq)$ is a complete lattice with joins $\vee$ and least element $\perp$. Let $f : V \to V$ be an arbitrary total function. Recall the definition of f's inflationary fixpoint as $\mathtt{ifp} f = \bigvee_\alpha \mathtt{ifp}^\alpha f$ over all ordinals α with

$$\begin{aligned}
\mathtt{ifp}^0 f &:= \perp \\
\mathtt{ifp}^{\alpha+1} f &:= \mathtt{ifp}^\alpha f \vee f(\mathtt{ifp}^\alpha f) \\
\mathtt{ifp}^\kappa f &:= \bigvee_{\alpha<\kappa} \mathtt{ifp}^\alpha f
\end{aligned}$$

where κ is a limit ordinal.

The fact that we want to embed MIC into a function calculus where functions are ordered by pointwise inclusion suggests to generalise the concept of an inflationary fixpoint slightly. Let $\mathtt{ifp}_x f$ for any $x \in V$ be defined as above with x instead of $\perp$. I.e. $\mathtt{ifp}_x f$ is the result of the inflationary fixpoint iteration starting in x rather than $\perp$. Thus, $\mathtt{ifp} f = \mathtt{ifp}_\perp f$. Then we can do a little bit of inflationary fixpoint arithmetic.

Lemma 1 For all $x \in V$, all total functions $f : V \to V$ we have: $\mathtt{ifp}_x f \geq x \vee \mathtt{ifp}_{x \vee f(x)} f$.

Proof We have $x = \mathtt{ifp}^0_x f \leq \mathtt{ifp}_x f$. Hence, only $\mathtt{ifp}_{x\vee f(x)} f \leq \mathtt{ifp}_x f$ remains to be shown. We will do this by trans-finite induction on the approximants. The base case is easy.

$$\mathtt{ifp}^0_{x\vee f(x)} f \;=\; x \vee f(x) \;=\; \mathtt{ifp}^1_x f \;\leq\; \mathtt{ifp}_x f$$

The case of a successor ordinal is not much harder.

$$\mathtt{ifp}^{\alpha+1}_{x\vee f(x)} f \;=\; \mathtt{ifp}^{\alpha}_{x\vee f(x)} \vee f(\mathtt{ifp}^{\alpha}_{x\vee f(x)}) \;\leq\; \mathtt{ifp}_x f \vee f(\mathtt{ifp}_x f) \;=\; \mathtt{ifp}_x f$$

by the hypothesis and because $\mathtt{ifp}_x f$ is indeed a fixpoint of f for any x.

Finally, the case of a limit ordinal does not pose a great deal of difficulty either.

$$\mathtt{ifp}^{\kappa}_{x\vee f(x)} f \;=\; \bigvee_{\alpha<\kappa} \mathtt{ifp}^{\alpha}_{x\vee f(x)} \;\leq\; \bigvee_{\alpha<\kappa} \mathtt{ifp}_x f \;=\; \mathtt{ifp}_x f$$

This finishes the claim. ■

Lemma 2 For all $f : V \to V$, all $x \in V$ and all finite ordinals α we have: $\mathtt{ifp}^{\alpha+1}_x f = \mathtt{ifp}^{\alpha}_{x\vee f(x)} f$.

Proof Simply by induction on α. The base case is

$$\mathtt{ifp}^1_x f \;=\; \mathtt{ifp}^0_x f \vee f(\mathtt{ifp}^0_x f) \;=\; x \vee f(x) \;=\; \mathtt{ifp}^0_{x\vee f(x)} f$$

The step case is

$$\begin{aligned}\mathtt{ifp}^{\alpha+2}_x f \;&=\; \mathtt{ifp}^{\alpha+1}_x f \vee f(\mathtt{ifp}^{\alpha+1}_x f)\\ &=\; \mathtt{ifp}^{\alpha}_{x\vee f(x)} f \vee f(\mathtt{ifp}^{\alpha}_{x\vee f(x)} f) \;=\; \mathtt{ifp}^{\alpha+1}_{x\vee f(x)}\end{aligned}$$

using the hypothesis. ■

We remark that Lemma 2 is not true for ordinals beyond ω. This should also be intuitively clear. Iterating for $\omega + 10$ steps starting in x for instance is not the same as iterating $\omega + 9$ steps when starting in $x \vee f(x)$. This simply reflects non-commutativity of ordinal addition.

Now note that $(V \to V, \sqsubseteq)$ forms a complete lattice too. Its order is defined pointwise: $f \sqsubseteq g$ iff $f(x) \leq g(x)$ for all $x \in V$. Its bottom element is denoted $\perp$ as well, and we use $\sqcup$ for its joins.

We associate with every total function $f : V \to V$ a functional $F_f : (V \to V) \to (V \to V)$ defined as $F_f(g) := \lambda x.x \vee g(x \vee f(x))$. Note that this is monotonic w.r.t. $\sqsubseteq$ since its argument g only occurs positively in the definition. According to the Knaster-Tarski Theorem [Tar55], it possesses a least fixpoint μF_f of type $V \to V$. We now claim that this can be used to find the inflationary fixpoint of f – provided that the inflationary fixpoint iteration stabilises after a finite number of iterations.

Lemma 3 Let $f : V \to V$ be a total function on a complete lattice V. Then $\mathtt{ifp} f \geq (\mu F_f)(\bot)$.

PROOF We will show a more general relation: for all $x \in V$ we have $\mathtt{ifp}_x f \geq (\mu F_f)(x)$. This statement about a relation in V can be lifted to a statement about a relation in $V \to V$ because the order in there is pointwise: $\lambda x.\mathtt{ifp}_x f \sqsupseteq \mu F_f$. Now this can be shown by fixpoint induction. According to the Knaster-Tarski theorem it suffices to show that $\lambda x.\mathtt{ifp}_x f$ is a pre-fixpoint of F_f which amounts to showing

$$\lambda x.\mathtt{ifp}_x f \;\sqsupseteq\; \lambda x.x \vee (\lambda x.\mathtt{ifp}_x f)(x \vee f(x)) \;=\; \lambda x.x \vee \mathtt{ifp}_{x \vee f(x)} f$$

Again, by the pointwise definition of $\sqsupseteq$ we need to show that for all $x \in V$: $\mathtt{ifp}_x f \geq x \vee \mathtt{ifp}_{x \vee f(x)} f$. But this is exactly what has been proved in Lemma 1. ■

Lemma 4 Let $f : V \to V$ be a total function on a complete lattice V. If $\mathtt{ifp} f = \mathtt{ifp}^\omega f$ then $\mathtt{ifp} f \leq (\mu F_f)(\bot)$.

PROOF Again, we generalise the statement. We will show for all $x \in V$ and all finite ordinals α: $\mathtt{ifp}^\alpha_x f \leq (\mu F_f)(x)$. The base case of $\alpha = 0$ is

$$\begin{aligned}
\mathtt{ifp}^0_x f &= x \\
&= x \vee \bot \\
&= x \vee (\mu^0 F_f)(x \vee f(x)) \\
&= (\mu^1 F_f)(x) \;\leq\; (\mu F_f)(x)
\end{aligned}$$

The case of the successor ordinals uses Lemma 2 and the fact that $(\mu F_f)(x) = x \vee (\mu F_f)(x \vee f(x))$ by unfolding.

$$\begin{aligned}
\mathtt{ifp}^{\alpha+1}_x f &= \mathtt{ifp}^\alpha_{x \vee f(x)} f \\
&\leq x \vee \mathtt{ifp}^\alpha_{x \vee f(x)} f
\end{aligned}$$

$$\leq\; x \vee (\mu F_f)(x \vee f(x)) \;=\; (\mu F_f)(x)$$

Finally, if $\mathtt{ifp} f = \mathtt{ifp}^\omega f$ then we have

$$\mathtt{ifp} f \;=\; \mathtt{ifp}^\omega f \;=\; \bigvee_{\alpha<\omega} \mathtt{ifp}^\alpha f \;\leq\; \bigvee_{\alpha<\omega} (\mu F_f)(\bot) \;=\; (\mu F_f)(\bot)$$

which finishes the claim. ■

Proof of Thm. 7.3: MIC $\subsetneq$ HFL^1 on finite structures.

As said before, the strictness is a consequence of the fact that there are FLC-definable properties (of finitely branching structures) that are provably not MIC-definable, and everything FLC-definable is also HFL^1-definable. What remains to be seen is that every MIC-definable property is also HFL^1-definable.

We first observe that adding simultaneous fixpoint operators does not increase the expressive power of HFL^1. The simultaneous fixpoint formula

$$\mu_i \begin{pmatrix} Z_1 & . & \varphi_1(Z_1,\ldots,Z_n) \\ & \vdots & \\ Z_n & . & \varphi_n(Z_1,\ldots,Z_n) \end{pmatrix}$$

abbreviates in the usual manner the formula

$$\mu Z_i.\varphi_i(\mu Z_1.\varphi_1(Z_1,\ldots,Z_i,\ldots),\ldots,Z_i,\ldots,\mu Z_n.\varphi_n(\ldots,Z_i,\ldots,Z_n))$$

etc. For better readability we omit type annotations here. They can easily be derived from the corresponding types in the simultaneous version of this formula.

Let $\varphi(X_1,\ldots,X_k) \in$ MIC. W.l.o.g. we can assume all inflationary fixpoint variables in φ to be distinct – not just the free ones $X_1,\ldots,X_k$. We inductively define an HFL^1 formula $\|\varphi\|$ of type Pr with free variables $X_1,\ldots,X_k$ of type Pr, s.t. for all LTS $\mathcal{T}$, all their states s, and all environments ρ:

$$\mathcal{T}, s \models_\rho \varphi \quad\text{iff}\quad \mathcal{T}, s \models_\rho \|\varphi\|$$

The translation is straight-forward for all cases other than those of an inflationary fixpoint operator.

$$\begin{aligned} \|q\| &:= q \quad \text{for all } q \in \mathcal{P} \\ \|\varphi \vee \psi\| &:= \|\varphi\| \vee \|\psi\| \end{aligned}$$

$$\begin{aligned}\|\neg\varphi\| &:= \neg\|\varphi\| \\ \|\langle a\rangle\varphi\| &:= \langle a\rangle\|\varphi\|\end{aligned}$$

It should be clear that the correctness property proposed above is an inductive invariant in these cases.

Now suppose $\varphi = \mathtt{ifp}_i(X_1, \ldots, X_n).(\varphi_1, \ldots, \varphi_n)$ for some $i \in \{1, \ldots, n\}$. Define $tr(\varphi)$ as $(\mu_i\mathcal{F})\ \underbrace{\mathtt{ff}\ \ldots\ \mathtt{ff}}_{n \text{ times}}$ with

$$\mathcal{F} \ := \begin{pmatrix} F_1 & . & \lambda X_1 \ldots \lambda X_n . X_1 \vee \big(F_1\ (X_1 \vee \|\varphi_1\|)\ \ldots\ (X_n \vee \|\varphi_n\|)\big) \\ & \vdots & \\ F_n & . & \lambda X_1 \ldots \lambda X_n . X_n \vee \big(F_n\ (X_1 \vee \|\varphi_1\|)\ \ldots\ (X_n \vee \|\varphi_n\|)\big) \end{pmatrix}$$

All X_i are of type Pr, and all F_i are of type $\mathrm{Pr}^0 \to \mathrm{Pr}$ – since the X_i can occur both positively and negatively in the $\|\varphi_j\|$.

Simply abbreviate $\rho[X_1 \mapsto S_1, \ldots, X_n \mapsto S_n]$ as ρ'. Note that $\|\varphi\| = (\mu F_f)(\bot, \ldots, \bot)$ where

$$f(S_1, \ldots, S_n) = \Big(\llbracket\, \varphi_1(X_1, \ldots, X_n)\,\rrbracket^{\mathcal{T}}_{\rho'}, \ldots, \llbracket\, \varphi_n(X_1, \ldots, X_n)\,\rrbracket^{\mathcal{T}}_{\rho'}\Big)$$

Hence, correctness of this case follows from the inductive hypothesis and Lemmas 3 and 4, as well as the fact that on finite structures, fixpoints of inflationary iterations are trivially reached after at most ω many steps.