EXTREMAL FIXPOINTS FOR HIGHER-ORDER MODAL LOGIC

Dissertation zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
im Fachbereich Elektrotechnik/Informatik der Universität Kassel

Florian Bruse

Oktober 2018

Erster Prüfer:	Prof. Dr. Martin Lange
Zweiter Prüfer:	Prof. Dr. Matthew Hague
Datum der Disputation:	23.11.2018

Zusammenfassung

Die vorliegende Dissertation untersucht das Zusammenspiel extremaler Fixpunkte mit Konstrukten höherer Ordnung am Beispiel der modalen Logik höherer Ordnung (HFL). Dabei handelt es sich um eine Erweiterung des bekannten modalen μ-Kalküls um einen einfach getypten Lambda-Kalkül. Die entstehende Logik ist sehr ausdrucksstark und das Zusammenspiel der einzelnen Bestandteile ist bisher wenig untersucht worden. Ziel der Arbeit war es, dieses Zusammenspiel zu charakterisieren.

Eine erste Charakterisierung liegt vor, indem die denotationale Semantik von HFL in eine äquivalente operationale Semantik übersetzt wurde. Das Ergebnis sind die sogenannten alternierenden Paritäts-Krivine-Automaten (APKA), welche eine Erweiterung des automatentheoretischen Gegenstücks des μ-Kalküls, der Paritäts-Automaten, darstellen. Die passende Erweiterung geschieht durch die Übernahme des Verhaltens der sogenannten Krivine-Maschine, welche Normalformen für den einfach getypten Lambda-Kalkül berechnet. Das Modell der APKA stellt das erste operationale Modell dar, welches über die Klasse aller Strukturen äquivalent zu HFL ist, was auch bewiesen wird. Von besonderer Schwierigkeit war es, die Akzeptanzbedingung korrekt zu wählen. Die für Fixpunktlogiken naheliegende Paritätsbedingung ist ohne Weiteres nicht einsetzbar. Stattdessen wird die Akzeptanzbedingung über eine zusätzlichen Hilfsstruktur ausgewertet, welche unendliche Fixpunktrekursion von Nebeneffekten der Konstrukte höherer Ordnung trennt. In Vorbereitung auf die Charakterisierung von HFL durch APKA wird noch ein Model-Checking-Spiel angegeben, welches zwar auch die Semantik von HFL korrekt erfasst, allerdings im Gegensatz zu APKA einen im Allgemeinen unendlichen Zustandsraum hat.

Ein zweites Thema der Arbeit war das Verhalten extremaler Fixpunkte, wenn die Interaktion dieser Fixpunkte mit den Effekten höherer Ordnung beschränkt wird. Dabei wurde zunächst das sogenannte endrekursive Fragment von HFL untersucht, über welchem es weniger aufwändig ist, die Modelleigenschaft einer Struktur relativ zu einer Formel zu überprüfen. Für dieses Fragment wird ein Model-Checking-Algorithmus angegegeben und durch eine passende untere Schranke Optimalität bewiesen.

Weiterhin spielten Fragen nach der Striktheit der Fixpunkt-Alternierungshierarchie eine Rolle. Neben der Angabe einer automatenbasierten, schlüssigen Definition dieser Hierarchien, die bisher auf syntaktischer Ebene nicht untersucht wurde, konnte für zwei Fragmente von HFL die Striktheit der Hierarchie bewiesen werden. Dies geschieht in Adaption eines Resultats von Arnold, in dem jeweils Läufe der jeweiligen Automaten über einem Baum selbst wieder als Baum kodiert werden. Eine Anwendung des Banach'schen Fixpunktsatzes erlaubt dann auf die Striktheit zu schließen. In einem weiteren Abschnitt wurden Situationen untersucht, in denen es möglich ist, die Polarität einer Fixpunktdefinition umzudrehen.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Verification of complex systems is a prime focus of research in contemporary theoretical computer science. A complex system can be a piece of software, a hardware design, or even a procedure, e.g., an election protocol or an emergency procedure. Verification means the automated certification with mathematical security that the system in question has a given property. A common approach is to translate the system into a mathematical structure, e.g., a transition system, and to specify the desired property in some kind of logical system. The problem of *model checking* is to decide if the structure encoding the system satisfies the formula, which is equivalent to the system having the property in question. This technique now sees widespread industrial use both in software verification and in chip design [27, 7, 32]; Clarke, Emerson, and Sifakis have been awarded the 2007 Turing Award for their contributions to the development of model checking.

In light of the importance of the technique of model checking, a lot of research has gone into cataloging and categorizing the properties of combinations of specification languages and classes of systems. Common questions are whether the model-checking problem of such a combination is decidable, i.e., whether it can be automated, and, if yes, how computationally expensive this is. Research into the necessary computational resources needed to solve the model-checking problem for a given combination then branches into the analysis of restricted classes and extensions. Additionally, one is interested in the question of satisfiability, i.e., the question whether, given some specification, there actually is a system that has the property specified.

Common specification languages in industrial use are *temporal* or *modal* logics, which make it possible to formulate assertions on the behavior of a system over time, e.g., the behavior of a program at runtime. In order to make these assertions, temporal logics are usually equipped with some kind of fixpoint constructor. A traditional "yardstick", although not necessarily in common use itself, is the *Modal μ-Calculus* ($\mathcal{L}_\mu$) [59]. It is obtained by extending Basic Modal Logic by (second-order) least- and greatest-fixpoint quantifiers. $\mathcal{L}_\mu$ subsumes many other commonly used logics such as LTL, CTL and CTL* [31]. The theory surrounding $\mathcal{L}_\mu$ is well understood; $\mathcal{L}_\mu$ also links nicely to the theory of automata developed around the question of decidability of the theory of Monadic Second Order Logic of infinite trees [22, 76]. In fact, $\mathcal{L}_\mu$ is equi-expressive to parity automata [34]. This connection is very useful both for practical problems, where $\mathcal{L}_\mu$ model checking and parity-automaton model-checking are considered two sides of the same coin, and for theoretical results, where it can be exploited to yield additional results regarding the necessity to nest extremal fixpoints [4, 61].

However, the expressive power of $\mathcal{L}_\mu$ is restricted to express regular properties due to its equivalence to regular automata and Monadic Second-Order Logic [47] over tree structures. Already very simple properties such as universality of nondeterministic finite automata or, equivalently, uniform inevitability in trees, cannot be expressed in $\mathcal{L}_\mu$ [33]. Moreover, any kind of property that requires unbounded counting, like buffer underflows or proper context-free properties such as well-nesting of calls and returns, is not expressible in $\mathcal{L}_\mu$. Thus, research into extensions of $\mathcal{L}_\mu$ beyond the bound of regularity has been a major effort in the last two decades. An early approach to this problem [43] extends Propositional Dynamic Logic with context-free operators. Recent results concern the formalization of the call and return structure of programs [2] and automaton models that circumvent the well-known pitfalls surrounding context-free properties [3, 66]. Another proposal lifts the semantics of $\mathcal{L}_\mu$ entirely to second order [70] by introducing Fixpoint Logic with Chop (FLC). This proposal already exhibits a property induced by its strong expressive power: The satisfiability problem becomes undecidable, i.e., it is not automatically verifiable whether a specification in this language can actually be fulfilled.

The next logical step in this escalation of expressive power is to add unrestricted higher-order features to $\mathcal{L}_\mu$. Viswanathan and Viswanathan propose Higher-Order Modal Fixpoint Logic (HFL) [90], which amalgamates $\mathcal{L}_\mu$ with a simply-typed lambda calculus. The resulting logic is naturally stratified by type-theoretic order and enjoys very high expressive power: Already low-order fragments can express a multitude [64] of commonly used properties. On the other hand, HFL inherits FLC's undecidability of the satisfiability problem, and the model checking problem for order-k formulas is k-EXPTIME-complete [6].

Another area of active research is $\mathcal{L}_\mu$ model-checking over infinite, yet finitely presented structures. A common way to encode such structures, for example the control graphs of functional programs, are Higher-Order Recursion Schemes (HORS) [54]. $\mathcal{L}_\mu$ model-checking of the trees generated by HORS is decidable [73, 56] and equivalent to $\mathcal{L}_\mu$ model-checking of the control graphs of Collapsible Pushdown Automata (CPDA) [42], a generalization of standard pushdown automata. Model checkers usable in practice are available for HORS model-checking [86, 78] and CPDA model-checking [15]. See [74] for a more thorough overview of the field.

A recently published pair of translations [55] between the $\mathcal{L}_\mu$ model-checking problem of HORS and the HFL model-checking problem links both worlds. These translations open up either side to technology import from the other side, which one expects would be beneficial for the theory of HFL model checking as the lesser developed one. However, there is also the expectation to generate new impulses for the $\mathcal{L}_\mu$ verification of higher-order infinite systems [57]. This motivates the study of the behavior of HFL, in particular, since it differs from the setting of HORS and CPDA model-checking in the following sense: The higher-order behavior of the latter two is contained exclusively in the structure of the system that is to be verified, while the extremal fixpoint behavior of the problem manifests itself in the $\mathcal{L}_\mu$ formula to be checked against, respectively the parity automaton it is translated into. On the other hand, in the context of HFL, both the higher-order behavior as well as extremal fixpoints are contained in the formula part of the problem, while the structure to be model checked is comparatively simple. Hence, it is of interest to understand how higher-order behavior interacts with extremal fixpoints if not separated into two parts of the input.

1.2 Contents of the Thesis

This thesis studies the interplay of extremal fixpoint constructors and higher-order constructs in the modal setting, i.e., in HFL. As a main result, we exhibit two different ways to give operational semantics to HFL, which, as a logic, comes with denotational semantics. The first such characterization is a new HFL model-checking game. It differs from the model-checking game exhibited in [6] in two ways: Fixpoint operators are handled natively via a μ-signature argument [84], as opposed to their elimination via unfolding in [6], and function application (as a manifestation of higher-order behavior) is handled via β-reduction. In [6], application is handled via a two-step alternating procedure where the play in question continues in the operator or the operand, but never both.

The second characterization of HFL, and a central piece of the thesis, is via an automaton model equivalent to HFL, namely *Alternating Parity Krivine Automata* (APKA). This model is an extension of the well-known parity automata by Krivine's Abstract Machine [60], a call-by-name computation model for the Simply-Typed Lambda Calculus. The acceptance condition uses an a posteriori condition called *unfolding trees*, which isolates infinite recursion in an infinite run of the automaton from side-effects introduced by higher-order behavior. Acceptance is then decided via an ordinary parity condition on said infinite recursion in the unfolding tree. This generalizes a technique used for FLC [62] to HFL; this rather complex condition is required due to the complexity of the interplay between extremal fixpoints and higher-order behavior. There are parallels to the winning condition in *Visibly Pushdown Games* [66] in that this condition isolates certain parts of the computation from the acceptance condition, however the structure of unfolding trees is more complicated than a simple stack structure. The correctness proof for this condition, as well as the pair of translations between HFL and APKA and their correctness proofs, are of according difficulty.

As a second aspect, and as an application of the first part, we study the behavior of HFL, respectively APKA, if the interaction between extremal fixpoints and higher-order constructs is restricted, either syntactically or due to the class of structures over which they are evaluated. A first contribution is the notion of *tail recursion*, which is the HFL equivalent of the corresponding notion in programming. Besides a restriction on how boolean alternation interacts with recursion, tail-recursive formulas are prohibited from containing recursive calls of fixpoint definitions on the operand side of an application, hence the name.

We show that the model-checking problem for tail-recursive formulas is easier by half an exponent, i.e., for tail-recursive formulas of order $k \geq 1$ it is $(k-1)$-EXPSPACE complete instead of k-EXPTIME complete [21, 20]. In particular, the completeness result shows that the restriction to tail recursion, while making the problem easier, does not make it trivial. The reason for this behavior is the fact that recursion of extremal fixpoints and higher-order behavior is almost completely decoupled in tail-recursive formulas. This becomes apparent in the context of so-called *simple* APKA, which share the restriction on operand-side recursion. Here, it can be seen that the complicated structure of unfolding trees degenerates to the recursive behavior encountered in $\mathcal{L}_\mu$.

We use this characterization to show strictness of the fixpoint alternation hierarchy for simple APKA, i.e., we show that, for this class, permitting more distinct priorities in an automaton strictly increases expressive power. We obtain the same result for order-1 HFL and APKA, extending the result in [62] from FLC to full order-1 HFL. Finally, we show that, at low type order, and over finite structures, where fixpoints stabilize after finitely many steps, fixpoint definitions of one polarity can be rewritten into equivalent ones of the opposite polarity, albeit at the cost of an increase in the order of the formula by one.

1.3 Structure of the Thesis

In Chapter 2, we introduce the necessary background from the literature, as well as established results. Besides standard objects such as orders and trees, we cover $\mathcal{L}_\mu$ and parity automata and give a short primer on the Simply-Typed Lambda Calculus. The rest of the chapter is dedicated to the exposition of HFL. In Chapter 3, we present an HFL model-checking game that we use later as an intermediate between the denotational semantics of HFL and APKA. In preparation of the model-checking game, we also establish a normal form for HFL formulas called *automaton normal form* and give a new proof that HFL admits negation normal form.

Chapter 4 contains the definition of APKA, as well as the proof that their acceptance condition is well-defined. The rest of the chapter is dedicated to a pair of translations between APKA and HFL, establishing that the former capture the semantics of the latter. In Chapter 5, we study the so-called tail-recursive fragment of HFL in which the interaction of fixpoints with the other operators in HFL is restricted. We present an algorithm to verify whether a formula is actually tail-recursive, a model-checking algorithm for tail-recursive formulas and a proof that the upper bound established by that algorithm is actually tight.

Chapter 6 contains results related to fixpoint alternation in the context of HFL and APKA. Besides a definition of alternation classes in terms of APKA, respectively their number of priorities, we study two fragments of the class of APKA where the acceptance condition is easier to manage than in the general case. We use this to show strictness of the alternation hierarchy when restricted to one of these automaton classes. Finally, we study certain settings in which it is possible to rewrite the polarity of a fixpoint definition into an equivalent one of the opposite polarity. Chapter 7 contains a summary of the work presented in this thesis, as well as an overview over tentative research targets.

1.4 Prior Publications

This thesis contains work from the following that has been published in part of completely.

The concept of APKA was introduced in [16] and refined in [17]; the version in Chapter 4 is a further development of this concept. The correctness proofs around the acceptance condition, as well as the translations from and to HFL are new. The unraveling technique in the translation from APKA to HFL follows an approach described for $\mathcal{L}_\mu$ in [18]. Chapter 5 contains the results published in [21] and submitted in [20]. The presentation here follows [20] closely and, in the case of Section 5.2, almost identically. The strictness result in Section 6.2.1 is a reworked result from [17], the results in Section 6.2.2 are new. Section 6.3 contains results from an unpublished workshop extended abstract [19]. Finally, the discussion of negation normal form in Section 3.1.2 follows an idea by Lozes [67]; the details of the construction and the correctness proof are new. The idea of automaton normal form was first sketched in [17]. The rest of Chapter 3 has not been published elsewhere, in particular the model-checking game.

Publications with contents that appear in this thesis in full or in part.

[16] Florian Bruse. Alternating parity krivine automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014

[17] Florian Bruse. Alternation is strict for higher-order modal fixpoint logic. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016.*, volume 226 of *EPTCS*, pages 105–119, 2016

[18] Florian Bruse, Oliver Friedmann, and Martin Lange. On guarded transformation in the modal μ-calculus. *Logic Journal of the IGPL*, 23(2):194–216, 2015

[19] Florian Bruse, Martin Lange, and Étienne Lozes. Collapses of fixpoint alternation hierarchies in low type-levels of higher-order fixpoint logic. unpublished

[20] Florian Bruse, Martin Lange, and Étienne Lozes. The complexity of model-checking the tail-recursive fragment of higher-order modal fixpoint logic. submitted

[21] Florian Bruse, Martin Lange, and Étienne Lozes. Space-efficient fragments of higher-order fixpoint logic. In Matthew Hague and Igor Potapov, editors, *Reachability Problems - 11th International Workshop, RP 2017, London, UK, September 7-9, 2017, Proceedings*, volume 10506 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017

1.5 Acknowledgments

First of all, I'd like to thank my advisor, Martin Lange. I thank him for teaching me a great many things both academic as well as non-academic in nature. I thoroughly appreciate the degree of both academic as well as personal freedom that comes attached with being Martin's Ph.D. student. I also like that Martin does not shy away from handing out responsibility if opportune, but never leaves one alone with it. In a nutshell: Martin treats you as an adult, which is very much appreciated.

I would also like to thank Matthew Hague for agreeing to serve as the second referee for my thesis. In particular, I thank him for doing so on quite the ambitious schedule, and for putting up with all the typos and missing parentheses in the referee version of this thesis.

I thank all the members of my group, both past and present, for the nice time we had. In chronological order, this is Bahareh, Étienne, Manuel, Milka, Norbert, Rüdiger, Maxime, Daniel and Lara, as well as Tina, who was always helpful when I needed something, and Michael. Special thanks go to Norbert for some very helpful tips during crunch time, and, of course, for all the political discussions. Although we disagree often, all these discussions made me see more clearly.

I thank Étienne Lozes for hosting me during my half-year stay in Cachan, France, and Georg Zetzsche and Stefan Göller for informally picking up his hosting role when Étienne moved to Nice unexpectedly.

Finally, I would like to thank my family. I would not be here without them, and I would not trade them for anyone. Stay who you are, and be safe!

Chapter 2

Preliminaries

In this chapter, we recall established results and notions from the literature, in particular in conjunction with modal logics and HFL. In the first section, we begin by fixing notation for the thesis. We then recall notions around orderings, in particular the order of the ordinals. We then look at graphs, in particular trees and DAGs. After that, we recall some general lattice theory, which leads to the Knaster-Tarski and Kleene characterizations of fixpoints of monotone functions over complete lattices. We also briefly sketch the Banach Fixpoint Theorem. The section continues with a primer on two-player semantic games and closes with a brief summary of the concepts we use related to decidability and complexity.

Section 2.2 revolves around the Modal μ-Calculus, Parity Automata and related concepts. In Section 2.3, we give a brief overview over the Simply-Typed Lambda Calculus and introduce Krivine's Abstract Machine. Section 2.4 contains the definition of Higher-Order Modal Fixpoint Logic (HFL), as well as syntactical notions around it and a summary of known properties of HFL.

2.1 General Mathematical Concepts

2.1.1 Notation

We consider 0 to be a natural number. Given natural numbers n and k, with 2^n_k we denote the following:

$$2^n_k = \begin{cases} n \text{ if } k = 0 \\ 2^{2^n_{k-1}} \text{ if } k > 0. \end{cases}$$

We say that a function $f\colon \mathbb{N} \to \mathbb{N}$ grows *nonelementarily* if it is not bounded by the function $n \mapsto 2^n_k$ for any $k \in \mathbb{N}$.

Throughout this thesis, we frequently deal with tuples where only some of the values are relevant in the specific context. In order to reduce clutter, in this case, we replace irrelevant entries by _ with the tacit assumption that the entries not displayed exist and are from the correct domains. For example, if it is clear from context that we are dealing with pairs (a, b) such that the natural number a divides the natural number b, we write $(5, \_)$ for a pair of the form $(5, 5n)$ if n is not important. Given a set S, we write $|S|$ to denote the cardinality of S. Given a function $f\colon A \to B$ and some subset $A' \subseteq A$ of A, we write $f \restriction A'$ to denote the restriction of f to A'. Given a relation $R \subseteq S^k$ of arity k and some subset $T \subseteq S$, we write $R \restriction T^k$ to denote the restriction of R to T^k.

2.1.2 Orders

Let S be a set. A binary relation $\leq \subseteq S^2$ is called a *partial order* if it is reflexive, antisymmetric and transitive, i.e., if

- for all $s \in S$ we have that $s \leq s$ holds,
- for all $s_1, s_2 \in S$ we have that $s_1 \leq s_2$ and $s_2 \leq s_1$ implies that $s_1 = s_2$, and
- for all $s_1, s_2, s_3 \in S$ it holds that $s_1 \leq s_2$ and $s_2 \leq s_3$ implies that $s_1 \leq s_3$.

$\leq$ is called *total* instead of partial if, moreover, for all $s_1, s_2 \in S$ at least one of $s_1 \leq s_2$ or $s_2 \leq s_1$ holds.

A binary relation $< \subseteq S^2$ is called a *strict partial order* if it is irreflexive and transitive, i.e., if

- for all $s \in S$, never $s < s$ holds and
- for all $s_1, s_2, s_3 \in S$ holds that $s_1 < s_2$ and $s_2 < s_3$ implies that $s_1 < s_3$.

A strict partial order $< \subseteq S^2$ is called *total* if, moreover, for all $s_1, s_2 \in S$ exactly one of $s_1 < s_2$, $s_1 = s_2$ and $s_2 < s_1$ holds.

We write $(S, \leq)$ to denote a structure with underlying set S where $\leq$ is a partial or total order. In this case, we often identify a partial order with its underlying set and just say that S is a *partially ordered set.* Sometimes, the order will be clear from context and not be given explicitly. Similarly, we say that S is a *totally ordered set* or a *total order*, if it can be equipped with such an order.

Every strict partial order $(S, <)$ can be made non-strict by adding all pairs of the form (s, s) to $<$, and a non-strict order can be made strict by removing all these pairs. Either direction preserves totality. Given a partial order $(S, \leq)$, a *chain* is a set $C \subseteq S$ such that the restriction of $\leq$ to C is total, i.e., any two elements of C are comparable with respect to $\leq$. The definition is the same for strict partial orders. A strict partial or total order $< \subseteq S^2$ is called *well-ordered* if there are no infinitely descending chains, i.e., if there is no sequence $(s_i)_{i \in \mathbb{N}}$ in S such that $s_{i+1} < s_i$ for all $i \in \mathbb{N}$.

Let $(S, \leq_S)$ and $(T, \leq_T)$ be two partial orders. A function $f\colon S \to T$ is called *monotone* if, for all $s_1, s_2 \in S$ such that $s_1 \leq_S s_2$, we have that $f(s_1) \leq_T f(s_2)$. The analogue definition holds for strict partial orders.

Symbols we use for orders are $\leq, \subseteq, \sqsubseteq$, etc. for non-strict orders and $<, \subsetneq, \prec$ etc. for strict orders. We use mirrored symbols to denote the converse of an order, i.e., given an order $\leq \subseteq S^2$, we write $s_1 \geq s_2$ to denote $s_2 \leq s_1$.

The Ordinals

One particular strict total order, in fact a well-order, that is important for this thesis is the ordinals, which intuitively generalizes and extends the order of the natural numbers induced by the usual interpretation of $<$. Notable properties the ordinals share with the naturals are the existence of a least element of every collection and closure under successors. The former means that, for each nonempty collection of ordinals, there is a a smallest ordinal equal to or smaller than all the ordinals in the collection. The latter means that, for each ordinal, there exists a strictly bigger ordinal. In fact, there is a smallest such ordinal.

However, the ordinals are not a set, which makes an easy definition hard to come by for two reasons: The standard way to define a relation by relating to another set or relation is not available, and, technically, the ordinals do not fit our definition of an order as a binary relation on a set. We circumvent the latter problem by assuming that the definitions around orders carry through to relations that are not over sets, but proper classes, and we are not going to give a formal definition. Instead, we refer the reader to the literature, (e.g., [48]), and list a number of properties of the class of the ordinals (Ord) that we are going to use:

- The ordinals form a strict total well-order,
- 0 is the least ordinal,
- every ordinal α has a a successor $\alpha + 1$, which is the smallest ordinal β such that $\alpha < \beta$,
- for every set[1] of ordinals there exists a supremum, i.e., a least ordinal bigger than all ordinals in the set, and
- the cardinality of any set is an ordinal.

An ordinal that is the successor of another ordinal, i.e., there is a unique greatest ordinal smaller than it, is called a *successor ordinal.* An ordinal that is not a successor ordinal and not 0 is called a limit ordinal. A notable ordinal is ω, the first limit ordinal, i.e., the supremum of the natural numbers.

2.1.3 Words, Graphs, Trees, and DAGs

Let S be a set. S^* denotes the set of all finite sequences of elements from S, i.e., all sequences of the form $s_1 \cdots s_n$ where $s_i \in S$ for all $1 \leq i \leq n$. Such a sequence is also called an *S-word*. The unique sequence of length 0 is called the *empty word* and is denoted by ε. Given a word $w = s_1 \cdots s_n$, any sequence of the form $s_1 \cdots s_m$ such that $m \leq n$ is called a *prefix* of w. Note that the empty word is a prefix of every word, including itself. A subset of S^* is called an S-language, and, in the context of words and languages, S is also called an alphabet.

Let S be a set. A *graph* with labels in S is a triple (V, E, Λ) where $\emptyset \neq V$, and, moreover, $E \subseteq V^2$ is a binary relation, and $\Lambda\colon V \to S$ is a labeling function. The elements of V are called *vertices* or *nodes*, and the pairs $(v_1, v_2) \in E$ are called *edges.* A finite sequence $v_1, \ldots, v_n$ of vertices such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq n-1$ is called a *path* of length $n-1$ from v_1 to v_n in G, and an infinite sequence $(v_i)_{i \in \mathbb{N}}$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$ is called an infinite path in G. If there is a path from v_1 to v_2 then we say that v_2 is *reachable* from s_1.

A graph (V, E, Λ) is called a *rooted directed acyclic graph*, or, from now on, just a *directed acyclic graph* (DAG) if

- there is a unique element $v_0 \in V$ called the *root* such that, for all $v \in V$ there is a path from v_0 to v,
- for all $v \in V$, every path from v to itself has length 0.

[1]Note that this does only hold for sets, not for arbitrary collections

Furthermore, a DAG (V, E, Λ) is called an (unranked) *tree* if, for every $v \in V$, there is exactly one path from v_0 to v. We frequently identify trees and DAGs with their underlying set. Given a DAG (V, E, Λ) and two nodes $t, u \in V$, we say that t is a *descendant* of u if there is a path of length at least 1 from u to t. In this case, u is an *ancestor* of t. Moreover, t is a *son* or *successor* of u if $(u, t) \in E$. In this case, u is the *father* of t. If t_1 and t_2 are distinct and both sons of u, then they are *siblings* of each other. A node without sons is called a *leaf*. Given a DAG (V, E, Λ) and a node $t \in V$, the *sub-DAG* induced by t is the triple $(V', E \restriction (V')^2, \Lambda \restriction V')$ where V' is the set of nodes that are reachable from t. It is easy to verify that this is a DAG again and that t is its root.

A *ranked tree* $\mathbb{T}$ with labels in S is presented as a tuple (V, Λ) where $\emptyset \neq V \subseteq \mathbb{N}^*$ is a set of words of natural numbers such that $\Lambda \colon V \to S$ is a labeling function and

- V is prefix closed in the sense that if $t \in V$ and u is a prefix of t then $u \in V$,
- V is leftwards closed in the sense that if $ti \in V$ then also $ti' \in V$ for all $i' < i$.

Note that necessarily $\varepsilon \in V$, and that such a pair (V, Λ) can be extended to a triple (V, E, Λ) via $E = \{(t, ti) \mid ti \in V\}$. It is not hard to verify that this triple satisfies the definition of an unranked tree where ε is the root. Given a ranked tree (V, Λ) and a node $t \in V$, the node $t0$, if it exists, is called the *leftmost* son of t, and a son ti is to the left of ti' if and only if $i < i'$. The *level* of a node t is the length of t. We also use level to refer to the entirety of nodes with a given length. For example, the zeroth level contains only the root.

Given a ranked tree (V, Λ) and a node $t \in V$, we say that the branching degree of t is i if $t(i-1) \in V$ but $ti \notin V$. The branching degree of a leaf is 0. If the branching degree of a node is 2, we also say that the branching at t is binary. The branching degree of a ranked tree is the supremum of the branching degrees of all its nodes. A tree is called fully infinite binary tree if its underlying set is $\{0, 1\}^*$.

Given two sequences t, u in the context of a ranked tree, the juxtaposition tu is to be understood as the concatenation of the two sequences. For example, if $t = 00$ and $u = 10$, then $tu = 0001$ is on the second node from the left on the fourth level.

Finally, trees, both ranked and unranked, as well as DAGs, can come in an unlabeled version, in which case the labeling function Λ is not important and, consequently, we do not display it.

The following is a somewhat specialized version of Kőnig's Lemma [58]. The original statement is slightly more general, but implies the version stated here, which is sufficient for the purposes of this thesis.

Theorem 2.1.1 (Kőnig). *Let $\mathbb{T} = (V, \Lambda)$ be an infinite ranked tree, i.e., a tree such that $|V| \geq \omega$. Moreover, let $\mathbb{T}$ be of finite branching degree. Then $\mathbb{T}$ contains an infinite path starting at the root, i.e., there is a sequence $(i_j)_{j \in \mathbb{N}}$ such that for all $j \in \mathbb{N}$ the word $i_0 \cdots i_j$ is contained in V.*

2.1.4 Lattices

Let $(L, \leq)$ be a partial order. We call $s \in L$ the *supremum* of a set $S \subseteq L$ if

- $x \leq s$ for all $x \in S$
- for all s' such that $x \leq s'$ for all $x \in S$, we have that $s' \geq s$.

In this case, we write $s = \bigsqcup S$. Similarly, $i \in L$ is called the *infimum* of a set $S \subseteq L$, written $i = \bigsqcap S$ if

- $x \geq i$ for all $x \in S$
- for all i' such that $x \geq i'$ for all $x \in S$, we have that $i' \leq i$.

In the case of a two element set $\{a, b\}$, we write $a \sqcap b$ and $a \sqcup b$ for $\bigsqcap\{a, b\}$ and $\bigsqcup\{a, b\}$.

A partial order $(L, \leq)$ is called a *lattice* if each finite subset of L has a supremum and an infimum. A lattice is called *complete* if each subset, finite or not, has a supremum and an infimum. A complete lattice $(L, \leq)$ necessarily has least and greatest elements $\bot_L = \bigsqcap L$ and $\top_L = \bigsqcup L$, or just $\bot$ and $\top$ if the lattice is clear from context. A lattice $(L, \leq)$ is called *distributive* if, for each $s_1, s_2, s_3 \in L$, we have $s_1 \sqcap (s_2 \sqcup s_3) = (s_1 \sqcap s_2) \sqcup (s_1 \sqcap s_2)$. A distributive lattice L is called a *boolean lattice* if, for each $s \in L$, there is a unique element $\overline{s}$ such that $x \sqcap \overline{x} = \bot_L$ and $x \sqcup \overline{x} = \top_L$. This element $\overline{s}$ is called the *complement* of s.

Example 2.1.2. Let S be a set. Then the powerset of S together with set inclusion forms the complete lattice $(2^S, \subseteq)$. Its least and greatest elements are $\emptyset$ and S, respectively. Such a lattice is also called the *powerset lattice* of S. In fact, $(2^S, \subseteq)$ is a boolean algebra and the complement $\overline{s}$ of a set $s \subseteq S$ is $S \setminus s$.

The following can also be proved by straightforward verification.

Observation 2.1.3. Given a lattice $(L, \leq)$ and a set S, the set of functions from S to L forms a complete lattice $(S \to L, \sqsubseteq)$ if ordered pointwise, i.e., for $f, g\colon S \to L$ we have $f \sqsubseteq g$ if for all $s \in S$ we have that $f(s) \leq g(s)$. If L is complete or boolean, then so is $(S \to L, \sqsubseteq)$, and the complement of a function $f\colon S \to L$ is defined as $\overline{f}\colon x \mapsto \overline{f(x)}$ in the latter case.

Given a lattice L, the *height* of L, denoted by $\mathsf{ht}(L)$ is defined as the cardinality of the longest (strictly) ascending chain in L. Note that this is necessarily an ordinal.

Lemma 2.1.4. *Given a set S of size n, the height of the powerset lattice is $n + 1$. Given a set S of size n and a lattice L of height m, the height of $(S \to L)$ is nm.*

Proof. The height of the powerset lattice of a set of size n is bounded from above by $n + 1$ since each chain member must contain at least one element that all previous sets in the chain do not contain. Moreover, a chain that actually exhausts this bound can be obtained by fixing some order of the set and then, starting from the empty set, adding one element in each step.

The height of a function lattice from S to L is bounded by nm, since each function in a chain must increase on at least one argument with respect to all its predecessors. A chain that exhausts this bound can be constructed in the same way as in the previous case. □

Further reading on lattices can be found in [39].

2.1.5 Fixpoints

Given a set S and function $f\colon S \to S$, a *fixpoint* of S is an element $s \in S$ such that $f(s) = s$. Not all functions have fixpoints. For example, $f\colon \mathbb{N} \to \mathbb{N}$ with $f\colon n \mapsto n{+}1$ has no fixpoints. In the context of this thesis, there are two situations in which fixpoints arise. The first setting is fixpoints of monotone functions on complete lattices, and the second setting, used in Chapter 6, is fixpoints of contractions on complete metric spaces.

Fixpoints of Monotone Functions over Complete Lattices

The following theorem, due to Knaster and Tarski [85], establishes that a monotone function on a complete lattice always has unique least and greatest fixpoints.

Theorem 2.1.5 (Knaster-Tarski). *Let $(L, \leq)$ be a complete lattice and let $f\colon L \to L$ be a monotone function. Then the set of fixpoints of f forms a complete lattice itself. In particular, f has a least fixpoint l and a greatest fixpoint g, and they satisfy the following equations:*

$$l = \bigsqcap \{x \in L \mid f(x) \leq x\}$$
$$g = \bigsqcup \{x \in L \mid f(x) \geq x\}$$

Example 2.1.6. Let $(L, \leq)$ be a lattice and let $a \in L$ be any element of L. Let f_a be defined via $f_a\colon s \mapsto s \sqcup a$. Then the set of fixpoints of f_a is the set $\{s \in L \mid a \leq s\}$. The least fixpoint of f_a is $\{a\}$, and the greatest fixpoint of f_a is $\top_L$.

Moreover, given a lattice, there is a way to obtain the least and greatest fixpoints of a monotone function that is, in general, more efficient than using Theorem 2.1.5. This is called the Kleene Fixpoint Theorem [53].

Theorem 2.1.7 (Kleene). *Let $(L, \leq)$ be a complete lattice and let $f\colon L \to L$ be a monotone function. Define, for each ordinal α, approximations $f_\perp^\alpha$ and $f_\top^\alpha$ via*

$$\begin{aligned} f_\perp^0 &= \perp_L & f_\top^0 &= \top_L \\ f_\perp^{\alpha+1} &= f(f_\perp^\alpha) & f_\top^{\alpha+1} &= f(f_\top^\alpha) \\ f_\perp^\alpha &= \bigsqcup\{f_\perp^\beta \mid \beta < \alpha\} & f_\top^\alpha &= \bigsqcap\{f_\top^\beta \mid \beta < \alpha\} \text{ if } \alpha \text{ is a limit ordinal.} \end{aligned}$$

Then there are ordinals α, β such that $f_\perp^\alpha$ is the least fixpoint of f and $f_\top^\beta$ is the greatest fixpoint of f.

Example 2.1.8. Let $\mathbb{T} = (V, E, \Lambda)$ be a tree such that Λ has range $\{a, b\}$. Let $(2^V, \subseteq)$ be the powerset lattice on $\mathbb{T}$ and let L_a be the set of nodes in $\mathbb{T}$ that are leaves labeled a. Let $f\colon V \to V$ be defined as $f(S) = \{t \mid \text{ex. } u \in S \text{ s.t. } (t, u) \in E\} \cup L_a$. Then the least fixpoint of f is the set of nodes in $\mathbb{T}$ that have a successor that is in L_a, and the greatest fixpoint of f is the set of all nodes of $\mathbb{T}$.

Fixpoints of Contractions on Complete Metric Spaces

The rest of this section introduces the complete metric space of fully infinite binary trees and the Banach Fixpoint Theorem for use in Chapter 6. A reader familiar with these can skip towards the next section.

$\mathbb{R}_{\geq 0}$ denotes the non-negative reals. Let S be a nonempty set. A function $d\colon S^2 \to \mathbb{R}_{\geq 0}$ is called a *metric* on S if

- for all $s_1, s_2 \in S$, we have that $d(s_1, s_2) = d(s_2, s_1)$,
- for all $s_1, s_2, s_3 \in S$, we have that $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$, and
- $d(s_1, s_2) = 0$ if and only if $s_1 = s_2$.

Intuitively, a metric is a measure of distance between two elements of S. A pair (S, d) where d is a metric on S is called a *metric space*.

Example 2.1.9. Let L be a nonempty set and let S be the set of fully infinite binary trees with labels in L. We say that two trees $\mathbb{T}_1, \mathbb{T}_2 \in S$ differ at level i if there is a sequence in $t = \{0,1\}^i$ such that the labeling of t in $\mathbb{T}_1$ is different from that in $\mathbb{T}_2$. Define $d\colon S^2 \to \mathbb{R}_{\geq 0}$ via

$$d(\mathbb{T}_1, \mathbb{T}_2) = \begin{cases} \quad 0 \text{ if } \mathbb{T}_1 = \mathbb{T}_2 \\ 2^{-i} \text{ if } i = \min\{i \mid \mathbb{T}_1, \mathbb{T}_2 \text{ differ at level } i\} \text{ otherwise.} \end{cases}$$

Then (S, d) is a metric space.

Let (S, d) be a metric space. We say that a sequence $(s_i)_{i\in\mathbb{N}}$ of elements in S is a *Cauchy sequence* if, for all $\varepsilon \in \mathbb{R}_{\geq 0}$ there is $n \in \mathbb{N}$ such that for all $m, m' > n$, we have that $d(s_m, s_{m'}) < \varepsilon$. A Cauchy sequence $(s_i)_{i\in\mathbb{N}}$ is said to *converge* to some point $s \in S$ if, for all $\varepsilon \in \mathbb{R}_{\geq 0}$, there is $n \in \mathbb{N}$ such that, for all $n' > n$ we have that $d(s_{n'}, s) < \varepsilon$. In this case, we say that s is a *limit point* of the sequence. It is straightforward to see that limit points of a Cauchy sequence are unique, whence we speak of the limit point of such a sequence. A metric space (S, d) is called *complete* if every Cauchy sequence of elements of S converges to a limit point in S.

Example 2.1.10. Let $L = \{a, b\}$ and let (S, d) be the metric space of fully infinite binary trees with labels in L from Example 2.1.9. Let $\mathbb{T}$ be the tree in S where every node is labeled by a, and let $\mathbb{T}_i$ be the tree where every node in $\{0,1\}^{i-1}$ is labeled by a and every other node is labeled by b.

Then $d(\mathbb{T}, \mathbb{T}_i) = 2^{-i}$ for all $i \in \mathbb{N}$ and $d(\mathbb{T}_i, \mathbb{T}_{i'}) = 2^{-\min(i,i')}$ for all $i, i' \in \mathbb{N}$. Moreover, the sequence $(\mathbb{T}_i)_{i\in\mathbb{N}}$ is Cauchy and converges to $\mathbb{T}$. Finally, it can be shown that (S, d) is complete.

Let (S, d) be a metric space. A function $f\colon S \to S$ is called a *contraction* if there is $C < 1$ such that, for all $s_1, s_2 \in S$, we have that $d(f(s_1), f(s_2)) \leq C \cdot d(s_1, s_2)$. The famous Banach Fixpoint Theorem [8] then states that any contraction on a complete metric space has a unique fixpoint:

Theorem 2.1.11 (Banach)**.** *Let (S, d) be a complete metric space and let $f\colon S \to S$ be a contraction. Then there is a unique $s \in S$ that is a fixpoint of f, i.e., $f(s) = s$. Moreover, for any $s' \in S$, the fixpoint s is the limit of the Cauchy sequence $s', f(s'), f^2(s'), \ldots$.*

2.1.6 Games and Strategies

We consider two-player games on graphs. The players are $\mathcal{V}$, who usually is considered to be female, and $\mathcal{S}$, who traditionally is male. We write $\mathcal{P}$ to denote either of them. A *game* between $\mathcal{V}$ and $\mathcal{S}$ consists of a game graph (V, E) which is partitioned into $V_\mathcal{V}$ and $V_\mathcal{S}$ and a *winning condition*, i.e., a partial function $c\colon V^\omega \to \{\mathcal{V}, \mathcal{S}\}$. Here, V^ω denotes the set of infinite sequences from V, i.e., the set of functions from $\mathbb{N}$ to V. Note that we do not require E to be total, i.e., games can contain vertices with no outgoing edges. A finite sequence $(v_i)_{i\leq n}$ of nodes in V is called a *position.* A *play* of the game is a finite or infinite sequence $(v_i)_{i\in I}$ of nodes in V such that $(v_i, v_{i+1}) \in E$. Given an infinite play $p = (v_i)_{i\in\mathbb{N}}$, we consider $\mathcal{V}$ the winner if $c(p) = \mathcal{V}$ and we consider $\mathcal{S}$ the winner if $c(p) = \mathcal{S}$. If $c(p)$ is undefined then neither player wins. Finite plays are awarded to one of the players only if the opposing player is stuck (see below).

A *play* from some starting position v is generated as follows: The initial play is the one-element sequence $v_0 = v$. Given a partial play $(v_i)_{i\leq n}$, one of the players extends the play to $(v_i)_{i\leq n+1}$. If $v_n \in V_\mathcal{V}$ then $\mathcal{V}$ picks a successor v_{i+1}, i.e., a node v_{n+1} such that $(v_n, v_{n+1}) \in E$, otherwise $v_n \in V_\mathcal{S}$ and $\mathcal{S}$ picks a successor v_{n+1} to extend the play. If the player that is supposed to extend the play is stuck because there is no successor available, they lose the game immediately.

A *strategy* for a play $\mathcal{P}$ is a function $s\colon V^*V_\mathcal{P} \to V$ such that for all $v \in V_\mathcal{P}$ and all $w \in V^*$, we have that $(v, s(wv)) \in E$. A play $(v_i)_{i\in I}$ is played by $\mathcal{P}$ according to a strategy s if, for each v_n such that $v_n \in V_\mathcal{P}$, we have that $v_{n+1} = s(v_0 \cdots v_n)$. A strategy for player $\mathcal{P}$ is *winning* if all plays played according to it are won by $\mathcal{P}$. If $\mathcal{P}$ has a winning strategy for a game from some initial position, we say that $\mathcal{P}$ *wins* the game from this position. A strategy s for $\mathcal{P}$ is called *positional* if, for all $w_1v_1, w_2v_2 \in V^*V_\mathcal{P}$ such that $v_1 = v_2$, we have that $s(w_1v_1) = s(w_2v_2)$. A game is called *determined* if, for each starting position, one of the players has a winning strategy. Note that this requires that the winning condition be defined on suitably many plays.

Example 2.1.12. Let $\mathbb{T}$ be a ranked tree with root ε such that the leftmost subtree of any node is finite. Let $E = \{(t, ti) \mid ti \in \mathbb{T}\} \cup \{(t, \varepsilon) \mid t \text{ is a leaf }\}$. Let $V_\mathcal{V}$ contain all nodes at odd levels, and let $V_\mathcal{S}$ contain all nodes at even levels. The winning condition awards those plays to $\mathcal{V}$ that visit the root infinitely often. Then $\mathcal{V}$ has a positional winning strategy by moving towards the leftmost son in nodes belonging to $V_\mathcal{V}$.

Parity Games

An important subclass of games is that of parity games. Parity games come with a *priority labeling* Δ, i.e., a function that associates to each node in the underlying graph a *priority*, i.e., a natural number from some finite set. The winning condition of a parity game is then defined as follows: an infinite play $(v_i)_{i\in\mathbb{N}}$ on a given graph induces a sequence of natural numbers $(\Delta(v_i))_{i\in\mathbb{N}}$. Let p be the highest priority that occurs infinitely often in this sequence, i.e.,

$$p = \limsup_{n\to\infty} (\Delta(v_i))_{i\in\mathbb{N}} = \max\left\{p \mid |\{i \mid \Delta(v_i) = p\}| = \infty\right\}.$$

$\mathcal{V}$ wins the play $(v_i)_{i\in\mathbb{N}}$ if p is even, and $\mathcal{S}$ wins otherwise. Parity games are always determined [40] and admit positional winning strategies. This means that every parity game is won by one of the players, and if a player has a winning strategy in a parity game from some starting position, they have a positional winning strategy from that starting position. Given a parity game, the set of nodes from which $\mathcal{V}$ wins can be computed in time $\mathcal{O}(m \cdot 2^n)$ where m is the size of E and n is the size of V [92]. However, the exponent often is closer to the number of priorities of the game rather than the number of vertices. This upper bound has since been improved [50, 81] to, e.g., $\mathcal{O}(m * n^{(d/2)})$ where d is the number of priorities. Recently, new, quasi-polynomial algorithms for the problem of parity game solving have been proposed [23, 51, 65]. Quasi-polynomial in this context means a complexity in time $\mathcal{O}(n^{\log m+6})$ [23]. For a survey of parity games in practice, see [37, 87].

Example 2.1.13. The game from Example 2.1.12 can be considered as a parity game via $\Delta(\varepsilon) = 2$ and $\Delta(t) = 1$ for $t \neq \varepsilon$.

Parity games play an important role in this thesis in the sense that we investigate generalizations of the concept of a parity game in Section 2.4.5.

2.1.7 Decidability and Complexity

We choose not to introduce Turing Machines formally since their only purpose in this thesis is to define time- and space-bounded complexity classes. A classic source is [75].

A *decision problem* P consists of an alphabet Σ and a subset $I \subseteq \Sigma^*$, where I is called the set of *positive instances* of the problem. A decision problem is *decidable* if there is a Turing Machine with input alphabet Σ that halts on all inputs and accepts exactly those inputs that are in I. A decision problem is *semi-decidable* if there is a Turing Machine with input alphabet Σ that halts and accepts on all positive instances, i.e., on all inputs in I.

The Turing Machines we are interested in here are *deterministic* or *alternating.* Both kinds can simulate each other, but a deterministic machine simulating an alternating one generally needs more resources.

A decision problem is in $\mathsf{DTIME}(f(n))$ for some function $f \colon \mathbb{N} \to \mathbb{N}$ if there is a deterministic Turing Machine that decides it and halts after at most $\mathcal{O}(f(n))$ steps on all inputs of length n. A decision problem is in $\mathsf{DSPACE}(f(n))$ if there is a deterministic Turing Machine that decides it and which uses at most $\mathcal{O}(f(n))$ tape cells on all inputs of length n.

For $k \geq 0$, we define

$$k\text{-EXPTIME} = \bigcup_{m \in \mathbb{N}} \mathsf{DTIME}(2_k^{n^m})$$

and

$$k\text{-EXPSPACE} = \bigcup_{m \in \mathbb{N}} \mathsf{DSPACE}(2_k^{n^m}).$$

We also write PTIME for 0-EXPTIME and PSPACE for 0-EXPSPACE. By the Time Hierarchy Theorem [44], k-EXPTIME $\subsetneq$ $(k+1)$-EXPTIME for all $k \geq 0$, and by the Space Hierarchy Theorem [82], k-EXPSPACE $\subsetneq$ $(k+1)$-EXPSPACE for all $k \geq 0$.

2.2 The Modal μ-Calculus and Tree Automata

2.2.1 Labeled Transition Systems

Fix a set A of *actions* and a set $\mathcal{P}$ of *propositions*. A *labeled transition system* (LTS) with actions in A and propositions in $\mathcal{P}$ is a triple $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ where S denotes the underlying set of *vertices*[2], $(\xrightarrow{a} \mid a \in A) \subseteq S \times A \times S$ and $\mathcal{L}\colon S \to 2^{\mathcal{P}}$ denote the interpretation of the binary and unary relations in $\mathcal{T}$. Since A and $\mathcal{P}$ usually can be derived from $(\xrightarrow{a} \mid a \in A)$ and $\mathcal{L}$, we generally do not explicitly introduce them when presenting an LTS.

We write $v \xrightarrow{a} v'$ if $(v, a, v') \in \xrightarrow{a}$, we write $\mathcal{T}, v \models P$ to denote that $P \in \mathcal{L}(v)$, and we write $\mathcal{T}, v \not\models P$ to denote that $P \notin \mathcal{L}(v)$. We write $v \in \mathcal{T}$ to denote that $v \in S$. Finally, when referring to an LTS $\mathcal{T}$, it is tacitly assumed that it is of the form $(S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ unless explicitly said otherwise. Hence, any reference to a set S in the context of an LTS refers to its underlying set, and the same holds for $(\xrightarrow{a} \mid a \in A)$ and $\mathcal{L}$.

Sometimes it is necessary to distinguish a vertex in a transition system. A *pointed LTS* is a pair $\mathcal{T}, v_0$, where $\mathcal{T}$ is an LTS and $v_0 \in S$ is a distinguished vertex.

Example 2.2.1. Let $\mathcal{P} = \{P\}$ and $A = \{a\}$. Let $\mathcal{T} = (\mathbb{N}, \{(i, a, i+1) \mid i \in \mathbb{N}\}, \mathcal{L})$ where $\mathcal{L}(i) = \{P\}$ if $i = 2^m - 2$ for some $m \in \mathbb{N}$ and $\emptyset$ otherwise. Then $\mathcal{T}$ is an LTS with propositions in $\mathcal{P}$ and actions in A. Moreover, $\mathcal{T}, 0$ is a pointed LTS with distinguished vertex 0.

A tree $\mathbb{T} = (V, E, \Lambda)$ with labels in S naturally defines an LTS $\mathcal{T}_{\mathbb{T}}$ with one accessibility relation a and propositions in S. This LTS is defined as $(V, (\xrightarrow{a}), \mathcal{L})$ where $t_1 \xrightarrow{a} t_2$ holds if and only if $(t_1, t_2) \in E$, and where $\mathcal{L}(t) = \{\Lambda(t)\}$.

2.2.2 Bisimulation

We briefly sketch the bisimulation-related concepts needed in this thesis. See [12] for a more in-depth exposition.

Let A be a set of actions and let $\mathcal{P}$ be a set of propositions. Let $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ be an LTS with actions in A and propositions in $\mathcal{P}$. A *bisimulation relation* on $\mathcal{T}$ is a relation $R \subseteq S^2$ that satisfies the following properties:

- If $(v_1, v_2) \in R$ then, $\mathcal{L}(v_1) = \mathcal{L}(v_2)$,
- if $(v_1, v_2) \in R$ and $v_1 \xrightarrow{a} w_1$, then there is w_2 with $v_2 \xrightarrow{a} w_2$ such that $(w_1, w_2) \in R$,
- if $(v_1, v_2) \in R$ and $v_2 \xrightarrow{a} w_2$, then there is w_1 with $v_1 \xrightarrow{a} w_1$ such that $(w_1, w_2) \in R$.

We say that two vertices $v_1, v_2 \in \mathcal{T}$ are *bisimilar*, written as $v_1 \sim v_2$, if there is a bisimulation relation on $\mathcal{T}$ that contains the pair (v_1, v_2). It is well-known that $\sim$ is an equivalence relation on every LTS, and even a congruence with respect to $(\xrightarrow{a} \mid a \in A)$ and $\mathcal{L}$. Hence, for any LTS $\mathcal{T}$ there is a *bisimulation quotient* $\mathcal{T}/_{\sim}$ obtained as the factor of the LTS over $\sim$. The equivalence class of a vertex v in this factor structure is written as $[v]$.

[2]Usually, vertices or *nodes* in an LTS are called *states*. However, in order to reduce overlap with automaton states, we refer to the elements in an LTS as vertices or nodes.

Example 2.2.2. Consider the LTS $\mathcal{T} = (\mathbb{N}, \{(i, a, i+1) \mid i \in \mathbb{N}\}, \mathcal{L})$ where $\mathcal{L}(i) = \{P\}$ if i is even and $\emptyset$ otherwise. All the vertices of even numbers are mutually bisimilar, and all vertices of odd numbers are mutually bisimilar. The bisimulation quotient $\mathcal{T}/_\sim$ is then $\mathcal{T}/_\sim = (\{[0],[1],\}, \{([0], a, [1]), ([1], a, [0]), \mathcal{L})$ with $\mathcal{L}([0]) = \{P\}$ and $\mathcal{L}([1]) = \emptyset$.

Bisimulation can be extended to a relation between two pointed LTS. We say that $\mathcal{T}_1, v_1$ is *bisimilar* to $\mathcal{T}_2, v_2$, written as $\mathcal{T}_1, v_1 \sim \mathcal{T}_2, v_2$, if $v_1 \sim v_2$ in the disjoint union of $\mathcal{T}_1$ and $\mathcal{T}_2$. Any pointed LTS is bisimilar to one that is connected, obtained by restricting the LTS to the connected component of the distinguished vertex. Moreover, any pointed LTS $\mathcal{T}, v$ is bisimimlar to $\mathcal{T}/_\sim, [v]$.

We say that a formula φ in some logic is *bisimulation invariant* if it cannot distinguish bisimilar pointed LTS, i.e, if $\mathcal{T}_1, v_1 \sim \mathcal{T}_2, v_2$ entails that $\mathcal{T}_1, v_1 \models \varphi$ if and only if $\mathcal{T}_2, v_2 \models \varphi$. Note that this definition implicitly requires formulas to be interpreted in pointed LTS. A good example for formulas of this format are formulas of first-order logic with exactly one free variable. Not all of them are bisimulation invariant.

2.2.3 The Modal μ-Calculus

The Modal μ-Calculus ($\mathcal{L}_\mu$) is an extension of Basic Modal Logic by second-order monadic least- and greatest-fixpoint constructors. Its present form was presented by Kozen [59].

Syntax

Let A be a set of actions, and let $\mathcal{P}$ be a set of propositions. Additionally, let $\mathcal{X}$ be a set of (second order) fixpoint variables. φ is an $\mathcal{L}_\mu$ formula if it can be derived from the following grammar

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

where $P \in \mathcal{P}$, $a \in A$, and $X \in \mathcal{X}$. Note that we have chosen to present the syntax of $\mathcal{L}_\mu$ in *negation normal form* (NNF), i.e., such that negations are only allowed in front of atomic propositions. In another widely used definition unrestricted use of negation is permitted, but requires an additional condition on its use in the context of fixpoint variables and binders. This restriction is required in order to ensure the existence of fixpoints. On the other hand, one operator of the pairs $\vee$ and $\wedge$, $\langle a \rangle$ and $[a]$, and μ and ν can be removed in the presence of negation without losing expressive power. Since both definitions are equivalent and can be used interchangeably, the missing symbols in either of them can be added as syntactic sugar. We write σ to denote either of μ and ν. We also sometimes use `tt` and `ff`, which can be understood as abbreviations of $P \vee \neg P$, respectively $P \wedge \neg P$ for an arbitrary proposition P.

The set of subformulas of a given $\mathcal{L}_\mu$ formula, the set of free variables of a subformula, as well as its syntax tree and DAG, are defined as usual. A formal definition for HFL, of which $\mathcal{L}_\mu$ is a fragment, can be found in Definitions 2.4.2, 2.4.4, and 2.4.3. In order to avoid duplication, we do not formally state the equivalent notions here. The size of an $\mathcal{L}_\mu$ formula is defined as the size of the set of its subformulas.

An $\mathcal{L}_\mu$ formula is called *well-named* if it is closed and, for each fixpoint variable X, it contains exactly one subformula of the form $\sigma X.\, \varphi'$. Given a well-named $\mathcal{L}_\mu$ formula φ, there is a function fp_φ that associates the formula $\mathsf{fp}_\varphi(X) = \varphi'$ to each fixpoint variable X that occurs in φ, where φ' is from the unique subformula of the form $\sigma X.\, \varphi'$. We call $\sigma X.\, \mathsf{fp}_\varphi(X)$ the *defining formula* of X. Moreover, in a well-named $\mathcal{L}_\mu$ formula φ, the set of its fixpoint variables is partially ordered by $\succ$ which is defined as $X \succ Y$ if and only if $\mathsf{fp}_\varphi(Y)$ is a proper subformula of $\mathsf{fp}_\varphi(X)$. In a well-named formula, a variable X is called a *least-fixpoint variable* if it is bound by $\mu X.\, \varphi$, and it is called a *greatest-fixpoint variable* if it is bound by $\nu X.\, \varphi$.

We say that an $\mathcal{L}_\mu$ formula is in *Basic Modal Logic* (ML) if it can be derived from the following grammar:

$$\varphi ::= P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi$$

Clearly, every ML formula is an $\mathcal{L}_\mu$ formula. We define a notion of formula depth for ML formulas that measures the length of the longest path in the syntax tree of the respective formula.

Definition 2.2.3. For each ML formula φ, associate a number $\mathrm{depth}(\varphi)$ which is inductively defined via

$$\begin{aligned}
\mathrm{depth}(P) &= 1\\
\mathrm{depth}(\neg P) &= 1\\
\mathrm{depth}(\varphi_1 \vee \varphi_2) &= 1 + \max\{\mathrm{depth}(\varphi_1), \mathrm{depth}(\varphi_2)\}\\
\mathrm{depth}(\varphi_1 \wedge \varphi_2) &= 1 + \max\{\mathrm{depth}(\varphi_1), \mathrm{depth}(\varphi_2)\}\\
\mathrm{depth}(\langle a \rangle \varphi') &= 1 + \mathrm{depth}(\varphi')\\
\mathrm{depth}([a]\varphi') &= 1 + \mathrm{depth}(\varphi')
\end{aligned}$$

Semantics

We now define the semantics of $\mathcal{L}_\mu$ over a given LTS. Note that this requires that the sets of actions and propositions mentioned in a given $\mathcal{L}_\mu$ formula are present in the LTS, or, even better, that the respective sets coincide. Rather than requiring such a match every time we discuss the semantics of an $\mathcal{L}_\mu$ formula, we stipulate that we restrict ourselves to situations where the sets do match. We will also use this convention later for other HFL.

Let $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ be an LTS. An *interpretation* η is a partial function that assigns a subset of S to variables in $\mathcal{X}$. Given an interpretation η, we define the *update* $\eta[X \mapsto T]$ as the interpretation defined via

$$\begin{aligned}
&\eta[X \mapsto T](Y) = T && \text{if } Y = X\\
&\eta[X \mapsto T](Y) = \eta(Y) && \text{if } Y \neq X.
\end{aligned}$$

As a convention, we tacitly assume, when dealing with the semantics of a formula under an interpretation, that the interpretation defines values for all free variables of the formula. We do not display an interpretation for closed formulas.

The semantics $[\![\varphi]\!]^\eta_\mathcal{T}$ of an $\mathcal{L}_\mu$ formula over $\mathcal{T}$ and under η is defined inductively via

$$\begin{aligned}
[\![P]\!]^\eta_\mathcal{T} &= \{v \in S \mid P \in \mathcal{L}(v)\}\\
[\![\neg P]\!]^\eta_\mathcal{T} &= S \setminus \{v \in S \mid P \in \mathcal{L}(v)\}\\
[\![X]\!]^\eta_\mathcal{T} &= \eta(X)\\
[\![\varphi \vee \psi]\!]^\eta_\mathcal{T} &= [\![\varphi]\!]^\eta_\mathcal{T} \cup [\![\psi]\!]^\eta_\mathcal{T}\\
[\![\varphi \wedge \psi]\!]^\eta_\mathcal{T} &= [\![\varphi]\!]^\eta_\mathcal{T} \cap [\![\psi]\!]^\eta_\mathcal{T}\\
[\![\langle a\rangle\varphi]\!]^\eta_\mathcal{T} &= \{v \in S \mid \text{ s.t. ex. } v' \in [\![\varphi]\!]^\eta_\mathcal{T} \text{ with } v \xrightarrow{a} v'\}\\
[\![[a]\varphi]\!]^\eta_\mathcal{T} &= \{v \in S \mid \text{ s.t. f.a. } v' \text{ with } v \xrightarrow{a} v' \text{ we have } v' \in [\![\varphi]\!]^\eta_\mathcal{T}\}\\
[\![\mu X.\, \varphi]\!]^\eta_\mathcal{T} &= \bigsqcap\{T \subseteq S \mid [\![\varphi]\!]^{\eta[X\mapsto T]}_\mathcal{T} \subseteq T\}\\
[\![\nu X.\, \varphi]\!]^\eta_\mathcal{T} &= \bigsqcup\{T \subseteq S \mid T \subseteq [\![\varphi]\!]^{\eta[X\mapsto T]}_\mathcal{T}\}
\end{aligned}$$

Note that the semantics of $\mu X.\, \varphi$ and $\nu X.\, \varphi$ relative to an LTS and an interpretation are the least and greatest fixpoints of the operator $T \mapsto [\![\varphi]\!]^{\eta[X\mapsto T]}_\mathcal{T}$ (cf. Theorem 2.1.5).

Example 2.2.4. Let $G = (V, E, V_\mathcal{V}, V_\mathcal{S}, \Delta)$ be a parity game with priorities in $1, \dots, n$. Let $A = \{a\}$ and let $\mathcal{P} = \{N, U, P_1, \dots, P_n\}$. We construct an LTS $\mathcal{T}$ out of the game graph of G via $\mathcal{T} = (V, \{(v_1, a, v_2) \mid (v_1, v_2) \in E\}, \mathcal{L})$ where

$$\mathcal{L}(v) = \begin{cases} \{N, P_i\} \text{ if } v \in V_\mathcal{V} \text{ and } \Delta(v) = i\\ \{U, P_i\} \text{ if } v \in V_\mathcal{S} \text{ and } \Delta(v) = i \end{cases}.$$

Let φ be defined as

$$\sigma X_n.\, \dots \nu X_2.\, \mu X_1.\big(N \to \langle a\rangle \bigwedge_{1\le i\le n} (P_i \to X_i)\big) \wedge \big(U \to [a] \bigwedge_{1\le i\le n} (P_i \to X_i)\big).$$

Then $v \in [\![\varphi]\!]_\mathcal{T}$ if and only if $\mathcal{V}$ wins G from v [91].

This example shows that parity game solving reduces to $\mathcal{L}_\mu$ model-checking. The opposite also holds [34]. Since the translations take polynomial time and logarithmic space, both problems share the same complexity bounds.

There is also simple bottom-up global model-checking algorithm for $\mathcal{L}_\mu$ that combines equivalent algorithms for ML with the Kleene style characterization of fixpoints, and, hence, works along the inductive definition of the semantics of $\mathcal{L}_\mu$ [36]. Starting from leaf formulas, i.e., propositions or fixpoint variables, compute the semantics of each fixpoint formula in a bottom-up fashion. Fixpoint formulas are initially valued as the empty set or the full set, depending on whether the variable in question is a least or a greatest-fixpoint variable. If a formula with a fixpoint binder is reached, store the set computed and continue again from the leaves of this formula, but update the value of the associated variable to the set computed, and reset all sets associated to lower fixpoints with respect to $\succ$. If the fixpoint binder in question is reached again, compare the two sets. If they agree, continue further up, otherwise, repeat the iteration. Since the nesting depth of the fixpoint quantifiers is bounded by the size of the formula, this algorithm runs in polynomial space, but may require time exponential in the nesting depth of quantifiers.

2.2.4 Alternating Parity Automata

Fix a set of actions A and a set of propositions $\mathcal{P}$.

Definition 2.2.5. A (symmetric) Alternating Parity Automaton (PA) $\mathcal{A}$ is a four-tuple $(\mathcal{Q}, \Delta, Q_I, \delta)$ where

- $\mathcal{Q}$ is a finite nonempty set of (fixpoint) *states*,
- $\Delta\colon \mathcal{Q} \mapsto \mathbb{N}$ is a *priority labeling*,
- $Q_I \in \mathcal{Q}$ is the *initial state*, and
- δ is a *transition relation* mapping each fixpoint state from $\mathcal{Q}$ to an $\mathcal{L}_\mu$ formula derived from the grammar

  $$\varphi ::= P \mid \overline{P} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a\rangle\varphi \mid [a]\varphi \mid Q$$

  where $P \in \mathcal{P}$, $a \in A$ and $Q \in \mathcal{Q}$.

The *size* $|\mathcal{A}|$ of a PA $\mathcal{A}$ is defined as the size of the union of the subformula sets of its transition relation.

Note that other important measures of size are the number of states and the number of priorities, i.e., the cardinality of the range of Δ. In the transition relation, $\overline{P}$ stands for a negated proposition, i.e., for $\neg P$. We choose this notation to avoid having the negation symbol present in the formulas of the transition relation.

Acceptance

Given an LTS $\mathcal{T}, v_I$ and a PA $\mathcal{A} = (\mathcal{Q}, \Delta, Q_I, \delta)$, acceptance[3] of $\mathcal{A}$ from $\mathcal{T}, v_I$ is defined via a parity game. The game graph is the set V of pairs (v, φ) where $v \in \mathcal{T}$ and φ is a subformula of $\delta(Q)$ for $Q \in \mathcal{Q}$. The edge relation E and membership in $V_\mathcal{V}$ and $V_\mathcal{S}$ are defined depending on the components v and φ in an outgoing vertex:

- If φ is P or $\overline{P}$, then (v, φ) has no outgoing edges. Moreover, $(v, \varphi) \in V_\mathcal{V}$ if $\varphi = P$ and $\mathcal{T}, v \not\models P$ or if $\varphi = \overline{P}$ and $\mathcal{T}, v \models P$. Otherwise, $(v, \varphi) \in V_\mathcal{S}$.
- If φ is $\varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$, then (v, φ) has edges to (v, φ_1) and $v, \varphi_2)$ and belongs to $\mathcal{V}$ in the first case and to $\mathcal{S}$ in the second case.
- If $\varphi = \langle a\rangle\varphi'$ or $\varphi = [a]\varphi'$ then (v, φ) has edges to all pairs (w, φ') such that $v \xrightarrow{a} w$ and belongs to $\mathcal{V}$ in the first case and to $\mathcal{S}$ in the second case.
- If $\varphi = Q$, then (v, φ) has an outgoing edge to $(v, \delta(Q))$ and belongs to $\mathcal{V}$.

The priority labeling of the game assigns to pairs of the form (v, Q) the priority $\Delta(Q)$ and to all other pairs the priority i, where i is the least number such that $\Delta(Q) = i$ for $Q \in \mathcal{Q}$. The automaton $\mathcal{A}$ is said to *accept* the pointed LTS $\mathcal{T}, v_I$ if $\mathcal{V}$ wins the acceptance game from position (v_I, Q_I).

From the definition of acceptance, it is immediately clear that, without loss of generality, any PA with n priorities can be assumed to have priorities in $\{1, \ldots, n\}$ or $\{0, \ldots, n-1\}$.

[3]Note that, again, we tacitly assume that the sets of actions and propositions of the LTS and the PA in question match.

Example 2.2.6. Let $A = \{a\}$ and let $\mathcal{P} = \{P_1, P_2\}$. Let $\mathcal{A} = (\{Q_1, Q_2\}, \Delta, Q_2, \delta)$ be a PA with $\Delta(Q_2) = 2$ and $\Delta(Q_1) = 1$ and where

$$\delta(Q_2) = (P_2 \vee [a]Q_2) \wedge (\overline{P_2} \vee \langle a \rangle Q_1)$$
$$\delta(Q_1) = (\overline{P_1} \wedge [a]Q_2) \vee (P_1 \wedge \langle a \rangle Q_2).$$

On all LTS such that each vertex is labeled by exactly one of P_1 and P_2, this PA verifies that, from every vertex labeled by P_1, there is another vertex reachable labeled by P_2. Intuitively, $\mathcal{S}$ controls the transitions in Q_2 and can search for a vertex labeled by P_2. If such a vertex is reached, he can force the automaton to switch into Q_1, where $\mathcal{V}$ now searches for a vertex labeled by P_1. If $\mathcal{S}$ cannot find an offending vertex, the automaton either stays in Q_2 indefinitely, or returns to it infinitely often. Since Q_2 has priority 2, $\mathcal{V}$ wins such a play. However, if $\mathcal{V}$ is stuck in Q_1 indefinitely, she loses by the same argument.

PA enjoy easy complementation, i.e., given a PA $\mathcal{A}$, it is quite simple to obtain a PA $\overline{\mathcal{A}}$ which accepts a pointed LTS if and only if $\mathcal{A}$ does not. This complemented automaton is obtained from $\mathcal{A}$ by increasing every priority by 1 and replacing every operator in the transition relation by its opposite. This means replacing P by $\overline{P}$ and vice versa, switching $\wedge$ and $\vee$, and doing equally for diamonds and boxes.

Example 2.2.7. Let $\overline{\mathcal{A}} = (\{Q_1, Q_2\}, \Delta, Q_2, \delta)$ be a PA with $\Delta(Q_2) = 3$ and $\Delta(Q_1) = 2$ and where

$$\delta(Q_2) = (\overline{P_2} \wedge \langle a \rangle Q_2) \vee (P_2 \wedge [a]Q_1)$$
$$\delta(Q_1) = (P_1 \wedge \langle a \rangle Q_2) \wedge (\overline{P_1} \vee [a]Q_2).$$

This PA in fact accepts exactly the pointed LTS that the PA $\mathcal{A}$ from Example 2.2.6 does not.

It should be noted that alternating parity automata are often used in the context of ranked trees. In that context, the transition relation of the automaton depends on the arity of the node in question, and can distinguish different successors, respectively move to specific successors in specific states. Since ranked trees are not the focus of this thesis, we restrict ourselves to symmetric PA.

2.2.5 Translations Between $\mathcal{L}_\mu$ and PA

PA and $\mathcal{L}_\mu$ are equi-expressive, i.e.,

- for each PA $\mathcal{A}$, there is an $\mathcal{L}_\mu$ formula $\varphi_{\mathcal{A}}$ such that $\mathcal{A}$ accepts a pointed LTS $\mathcal{T}, v_I$ if and only if $v_I \in [\![\varphi_{\mathcal{A}}]\!]_{\mathcal{T}}$, and
- for each $\mathcal{L}_\mu$ formula φ, there is a PA $\mathcal{A}_\varphi$ such that $\mathcal{A}_\varphi$ accepts a pointed LTS $\mathcal{T}, v_I$ if and only if $v_I \in [\![\varphi]\!]_{\mathcal{T}}$.

We sketch the two translations that prove this equi-expressivity. This serves as a preparation of a similar pair of translations between HFL formulas and APKA (see Sections 4.3 and 4.4), which subsume $\mathcal{L}_\mu$, respectively PA. Although the former have much greater expressive power than the latter, the translations for HFL and APKA inherit both the basic approach and the necessity to unravel the respective automaton sketched below in the translation from PA to $\mathcal{L}_\mu$.

From $\mathcal{L}_\mu$ to PA

Let φ be a well-named $\mathcal{L}_\mu$ formula and let $\mathcal{X}$ be the set of its fixpoint variables. Without loss of generality, $\varphi = \sigma X \varphi'$, otherwise, replace φ by $\sigma X.\, \varphi$ where X does not occur in φ. Let $\mathcal{A}_\varphi = (\mathcal{Q}, \Delta, Q_I, \delta)$ be the PA where

- $\mathcal{Q} = \{Q_X \mid X \in \mathcal{X}\}$,
- $Q_I = Q_X$ if $\varphi = \sigma X.\, \varphi'$,
- $\delta(Q_X)$ is defined as follows: If $\mathsf{fp}_X(\varphi) = \sigma X.\, \psi$ then $\delta(Q_X) = \mathsf{to}_\delta(\psi)$ which is inductively defined as

$$\begin{aligned}
\mathsf{to}_\delta(P) &= P\\
\mathsf{to}_\delta(\neg P) &= \overline{P}\\
\mathsf{to}_\delta(\psi_1 \vee \psi_2) &= \mathsf{to}_\delta(\psi_1) \vee \mathsf{to}_\delta(\psi_2)\\
\mathsf{to}_\delta(\psi_1 \wedge \psi_2) &= \mathsf{to}_\delta(\psi_1) \wedge \mathsf{to}_\delta(\psi_2)\\
\mathsf{to}_\delta(\langle a\rangle\psi') &= \langle a\rangle\mathsf{to}_\delta(\psi')\\
\mathsf{to}_\delta([a]\psi') &= [a]\mathsf{to}_\delta(\psi')\\
\mathsf{to}_\delta(Y) &= Q_Y\\
\mathsf{to}_\delta(\sigma Y.\, \psi') &= Q_Y.
\end{aligned}$$

- $\Delta(Q)$ is defined inductively:
  - If X is a greatest-fixpoint variable and there is no Y such that $X \succ Y$, then $\Delta(Q_X) = 0$.
  - If X is a least-fixpoint variable and there is no Y such that $X \succ Y$, then $\Delta(Q_X) = 1$.
  - If X is a greatest-fixpoint variable and $\max\{\Delta(Q_Y) \mid X \succ Y\}$ is i, then $\Delta(Q_X) = i$ if i is even, otherwise $\Delta(Q_X) = i + 1$.
  - If X is a least-fixpoint variable and $\max\{\Delta(Q_Y) \mid X \succ Y\}$ is i, then $\Delta(Q_X) = i$ if i is odd, otherwise $\Delta(Q_X) = i + 1$.

It can be shown that $\mathcal{A}_\varphi$ is equivalent to φ in the sense described above. Moreover, the size of $\mathcal{A}_\varphi$ is linear in the size of φ.

From PA to $\mathcal{L}_\mu$

The reverse direction contains a new challenge: The precedence between different states in a PA is decided by the priority labeling. In an $\mathcal{L}_\mu$ formula, the precedence between two fixpoint variables is decided by their respective position in the syntax tree of the formula. The latter also restricts which fixpoint variables can occur in the defining formula of some fixpoint variable. On the other hand, any (fixpoint) state can occur in the transition relation of any other (fixpoint) state. Faithfully transferring the relations between the states of a PA into the syntax tree of an $\mathcal{L}_\mu$ formula requires additional attention.

Let $\mathcal{A} = (\mathcal{Q}, \Delta, Q_I, \delta)$ be a PA. Assume, for the sake of simplicity, that all states have a different priority[4]. For a set $\mathcal{R} \subseteq \mathcal{Q}$ and a state $Q \in \mathcal{Q}$, define the restriction of $\mathcal{R}$ to the states with priority at least $\Delta(Q)$ as

$$\mathcal{R} \upharpoonright Q = \{Q' \in \mathcal{R} \mid \Delta(Q') \geq \Delta(Q)\}.$$

Let $\mathcal{X} = \{X_Q^{\mathcal{R}} \mid \mathcal{R} \subseteq \mathcal{Q}, Q \in \mathcal{R}\}$ be a set of fixpoint variables. For each $Q \in \mathcal{Q}$, and each $\mathcal{R} \subseteq \mathcal{Q}$ that contains Q, define the formula $\varphi_Q^{\mathcal{R}}$ as $\sigma_Q X_Q^{\mathcal{R}}.\, \mathsf{toML}_{\mathcal{R}}(\delta(Q))$, where $\sigma_Q = \mu$ if $\Delta(Q)$ is even, and $\sigma_Q = \nu$ else, and where $\mathsf{toML}_{\mathcal{R}}(\delta(Q))$ is defined inductively as

$$\begin{aligned}
\mathsf{toML}_{\mathcal{R}}(P) &= P \\
\mathsf{toML}_{\mathcal{R}}(\overline{P}) &= \neg P \\
\mathsf{toML}_{\mathcal{R}}(\varphi_1 \vee \varphi_2) &= \mathsf{toML}_{\mathcal{R}}(\varphi_1) \vee \mathsf{toML}_{\mathcal{R}}(\varphi_2) \\
\mathsf{toML}_{\mathcal{R}}(\varphi_1 \wedge \varphi_2) &= \mathsf{toML}_{\mathcal{R}}(\varphi_1) \wedge \mathsf{toML}_{\mathcal{R}}(\varphi_2) \\
\mathsf{toML}_{\mathcal{R}}(\langle a \rangle \varphi') &= \langle a \rangle \mathsf{toML}_{\mathcal{R}}(\varphi') \\
\mathsf{toML}_{\mathcal{R}}([a] \varphi') &= [a] \mathsf{toML}_{\mathcal{R}}(\varphi') \\
\mathsf{toML}_{\mathcal{R}}(Q') &= X_{Q'}^{\mathcal{R} \upharpoonright Q'} \text{ if } Q' \in \mathcal{R} \\
\mathsf{toML}_{\mathcal{R}}(Q') &= \varphi_{Q'}^{\mathcal{R} \upharpoonright Q' \cup \{Q'\}} \text{ if } Q' \notin \mathcal{R}
\end{aligned}$$

The intuition here is that the nesting of the different fixpoint formulas correctly emulates the precedence induced by the priority labeling. This is achieved by constructing the formula as follows: If a fixpoint variable encoding a state with low priority were to occur freely in a formula encoding the transition relation of a state of higher priority, the variable encoding the state of low priority is bound again to a new fixpoint quantifier. Hence, occurrences of this variable (for the low priority state) in the defining formula of the variable encoding the high priority state do not refer to a fixpoint binder that occurs above that for the variable encoding the state of high priority in the syntax tree of the formula we construct. Otherwise, we would obtain incorrect semantics of the formula, i.e., the formula would not be equivalent to the automaton. This invariant is realized by annotating a set $\mathcal{R}$ of states to each variable binding via a superscript of the variable to be bound. This set of states denotes which variables may occur freely in the subformula to be constructed. At each binding, variables for states of lower priority than the state the bound variable is intended for are removed from this set, which makes sure that they are bound again as sketched above.

It can be shown that the above construction terminates and does not generate infinitely nested sequences of subformulas. Moreover, $\varphi_{\mathcal{A}} = \varphi_{Q_I}^{\{Q_I\}}$ can be shown to be equivalent to $\mathcal{A}$. The size of $\varphi_{\mathcal{A}}$ is at most exponential in the size of $\mathcal{A}$. This bound is tight, since examples for exponential blow-up exists. See [18] for a more in-depth analysis of the problem.

[4]This is, of course, a restriction. Its purpose is to have the space of states be totally ordered. This order can be replaced with a topological order respecting the partial order induced by the priorities. We use such a technique in the translation from APKA to HFL. See Section 4.4 for the general argument.

2.2.6 Fixpoint Alternation

The question whether it is actually necessary to nest least and greatest fixpoints in $\mathcal{L}_\mu$ formulas has garnered considerable interest. The reasons for this are twofold. The first reason is that $\mathcal{L}_\mu$ model-checking and PA acceptance checking are interreducible and, hence, both reduce to parity games. Since the complexity of deciding the winning set of a play in a parity game depends heavily on the number of priorities, which is inherited from the number of entangled least and greatest fixpoints, knowing whether the latter can be reduced has practical impact. The second reason is that the semantics of entangled least and greatest fixpoints tend to be hard to understand even for trained users, prohibiting industrial use of $\mathcal{L}_\mu$.

The first step in answering the question whether nested least and greatest fixpoints are necessary to retain expressive power is to define a measure on the entanglement. Syntactic characterizations for $\mathcal{L}_\mu$ formulas are available [72, 36], but not necessarily practical to use in proofs. An elegant characterization [72] of the alternation degree of a formula that actually contains recursion is available via the number and polarity of priorities of any equivalent PA.

Definition 2.2.8. Let $n > 0$. A PA is said to be in Σ_0^n if it is equivalent to one that has priorities in $\{1, \dots, n\}$ if n is even and priorities in $\{0, \dots, n-1\}$ if n is odd. It is said to be in Π_0^n if it is equivalent to one that has priorities in $\{1, \dots, n\}$ if n is odd and priorities in $\{0, \dots, n-1\}$ if n is even. An $\mathcal{L}_\mu$ formula is in Σ_0^n, respectively Π_0^n, if it is equivalent to a PA in Σ_0^n, respectively Π_0^n. An $\mathcal{L}_\mu$ formula or a PA is said to be in $\Sigma_0^0 = \Pi_0^0$ if it is equivalent to a formula in basic modal logic. A formula is *alternation-free* if, in the defining formulas of its least-fixpoint variables, no greatest-fixpoint variables appear freely, and vice versa.

The subscript 0 in the definition of these *alternation classes* refers to them being defined with respect to $\mathcal{L}_\mu$, as opposed to those defined in Definition 6.1.1 in Chapter 6. A definition in terms of automata for alternation-freeness is available via so-called *weak* automata [77]. The class of alternation-free formulas can be verified to be contained in both Σ_0^2 and Π_0^2.

The above alternation classes obviously satisfy a number of properties. For example, $\Sigma_0^n \subseteq \Sigma_0^{n+1}$ for all $n \geq 0$, and $\Sigma_0^n \subseteq \Pi_0^{n+1}$, and the equivalent inclusions hold for Π_0^n. Moreover, if an automaton is in Σ_0^n for some n, then its complement is in Π_0^n, and vice versa.

The above definition, together with the inclusions outlined, induces a hierarchy of the alternation classes. The questions above now reduce to the question whether this hierarchy of alternation classes is strict, or whether it collapses in the sense that some alternation class contains all $\mathcal{L}_\mu$ formulas. Moreover, this question can be relativized to a class of LTS by asking whether strictness holds over a given class of structures or not. For example, over any singleton class of (pointed) structures, every $\mathcal{L}_\mu$ formula is equivalent[5] to `tt` or `ff`. Research tends to focus on more interesting classes of structures, however.

There are a number of results on the problems outlined so far. It is known that, over finite structures without infinite paths, the alternation hierarchy for $\mathcal{L}_\mu$ collapses to Σ_0^1 and to Π_0^1, i.e., every formula is equivalent to one with only one kind of fixpoint [69]. Moreover, over the class of words, i.e., over the class of LTS

[5]This does not hold if one is interested in the set defined by the respective formulas.

where every vertex has at most one successor, the alternation hierarchy collapses to the alternation-free fragment [52], and the result has been generalized to a number of classes of LTS [41]. On the other hand, Bradfield has shown [13, 14] that the alternation hierarchy is strict over the class of fully infinite binary trees and, hence, over the class of all LTS. The original proof is quite involved and uses a reduction to the alternation hierarchy in first-order arithmetic with fixpoints [68]. However, the proof was later simplified considerably by Arnold [4] into a very elegant version using Banach's Fixpoint Theorem (Theorem 2.1.11).

Strictness According to Arnold

Since we use the pattern of Arnold's proof in Chapter 6, we briefly sketch it here.

Fix a set of propositions $\mathcal{P}_0^n = \{T, F, N, U, P_1, \ldots, P_n\}$. Given a PA $\mathcal{A}$ in Σ_0^n or Π_0^n and given a tree $\mathbb{T}$ with labels in $\mathcal{P}_0^n$, define a tree $T_0(\mathcal{A}, \mathbb{T})$ with labels in $\mathcal{P}_0^n$ that encodes the acceptance game of $\mathcal{A}$ over $\mathbb{T}$. Each position in the acceptance game induces a node in the tree, which is padded to be fully binary and infinite if necessary. The initial position (ε, Q_I) of the acceptance game induces the root. Depending on the kind of the second component of a configuration (u, φ) inducing a node t, the labeling of that the node and its successors is defined as follows:

- If φ is P or $\overline{P}$ then t is labeled by T if $\mathbb{T}, u \models P$, respectively if $\mathbb{T}, u \not\models P$, and is labeled by F otherwise. Both successors of t are also induced by (u, φ).
- If $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \varphi_1 \wedge \varphi_2$, then t is labeled by N, respectively by U. The left successor of t is induced by (u, φ_1) and the right successor is induced by (u, φ_2).
- If $\varphi = \langle a \rangle \varphi'$ of $\varphi = [a]\varphi'$, then t is labeled by N, respectively by U. The left successor of t is induced by $(u0, \varphi')$ and the right successor of t is induced by $(u1, \varphi')$.
- if $\varphi = Q$ with $\Delta(Q) = i$, then t is labeled by P_i if the priorities of $\mathcal{A}$ are in $\{1, \ldots, n\}$ and by P_{i+1} if the priorities of $\mathcal{A}$ are in $\{0, \ldots, n-1\}$. Both successors of t are induced by $(u, \delta(Q))$.

The intuition behind this encoding is that

- labeling by T or F signals that $\mathcal{V}$, respectively $\mathcal{S}$ wins the game encoded into the tree from the position in question,
- labeling by N or U signals that $\mathcal{V}$, respectively $\mathcal{S}$ chooses the next configuration from the position in question,
- labeling by P_i signals that a fixpoint configuration with priority i (if the priorities of $\mathcal{A}$ are in $\{1, \ldots, n\}$), respectively $i - 1$ (if the priorities of $\mathcal{A}$ are in $\{0, \ldots, n-1\}$) is reached in the acceptance game of $\mathcal{A}$ over $\mathbb{T}$.

Now consider the PA $\mathcal{A}_{n,0}^{\Sigma}$ and $\mathcal{A}_{n,0}^{\Pi}$ defined via $(\mathcal{Q}, \Delta, O, \delta)$ where

- $\mathcal{Q} = \{Q_1, \ldots, Q_n, O\}$,

- for $1 \leq i \leq n$, $\Delta(Q_i) = i$ for $\mathcal{A}^{\Sigma}_{n,0}$ if n is even and for $\mathcal{A}^{\Pi}_{n,0}$ if n is odd, respectively $\Delta(Q_i) = i - 1$ for $\mathcal{A}^{\Sigma}_{n,0}$ if n is odd and for $\mathcal{A}^{\Pi}_{n,0}$ if n is even, and $\Delta(O) = \Delta(Q_1)$ in either case,

- $\delta(Q_i) = Q_{i-1}$ for $1 < i \leq n$, $\delta(Q_1) = O$ and

$$\delta(O) = T \vee \Big(\overline{F} \wedge \big(\overline{N} \vee \langle a \rangle O\big) \wedge \big(\overline{U} \vee [a]O\big) \wedge \big(\overline{P_1} \vee \langle a \rangle Q_1\big) \wedge \cdots \wedge \big(\overline{P_n} \vee \langle a \rangle Q_n\big)\Big).$$

Note that the structure of this PA borrows heavily from Walukiewicz' formulas encoding the winning region in a parity game (cf. Example 2.2.4 [91]). Such an automaton then accepts the encoding $T_0(\mathcal{A}, \mathbb{T})$ of an acceptance game of an automaton of matching alternation class if and only if $\mathcal{V}$ wins this game. Intuitively, $\mathcal{V}$ can choose the successor of a node in the game for $\mathcal{A}^{\Sigma}_{n,0}$, respectively $\mathcal{A}^{\Pi}_{n,0}$ such that the nodes traversed induce a play of $\mathcal{A}$ over $\mathbb{T}$. Since the former automaton visits a configuration containing Q_i if and only if the node it is currently at is labeled by P_i, the sequence of priorities encountered in the play of $\mathcal{A}^{\Sigma}_{n,0}$, respectively $\mathcal{A}^{\Pi}_{n,0}$ corresponds to the sequence of priorities encountered by $\mathcal{A}$.

The final step of the proof consists in the observation that traversal from the acceptance game of $\mathcal{A}$ over $\mathbb{T}$ to the acceptance game of $\mathcal{A}^{\Sigma}_{n,0}$, respectively $\mathcal{A}^{\Pi}_{n,0}$ over $T_0(\mathcal{A}, \mathbb{T})$ forms a contraction on the complete metric space of fully infinite binary trees. The reason for this is that the acceptance game for the latter automaton lags behind the acceptance game for the former since it has to go through all the coding machinery. It follows that, by the Banach Fixpoint Theorem, the mapping $f_{\mathcal{A}} \colon \mathbb{T} \mapsto T_0(\mathcal{A}, \mathbb{T})$ has a unique fixpoint for each automaton of a matching alternation class. Some additional reasoning leads the assumption that $\mathcal{A}^{\Sigma}_{n,0} \in \Pi^n_0$ to a contradiction, which establishes that $\Sigma^n_0 \not\subseteq \Pi^n_0$. Symmetric proofs show the opposite non-inclusion. See Theorems 6.2.11 and 6.2.29 for the full argument in action.

Remark 2.2.9. Arnold's approach to showing strictness of a fixpoint alternation hierarchy crucially relies on the fact that the winning condition of the acceptance game of the automaton class in question can be stored to a sufficient degree in the encodings of game trees of the automaton class in question. See Examples 6.2.13 and 6.2.14 for cases in which this approach fails. In this context, it is important that the failure of the approach does not signal a collapse of the alternation hierarchy, just inadequacy of the approach.

2.3 The Simply-Typed Lambda Calculus

The Lambda Calculus is one of the oldest models of computation and was introduced by Church [25] in order to investigate the theory of computation. It can be understood as some kind of reduction system. The simply-typed version of the Lambda Calculus also goes back to Church [26]. For both versions, there is an abundance of theory, most of which is not relevant to this thesis. We restrict ourselves to the necessary prerequisites. A good source for further reading on the typed Lambda Calculus is [9].

Figure 2.1: The typing rules for λ_{ML}.

$$\frac{}{\Sigma \vdash P\colon \circ} \qquad \frac{\Sigma \vdash t_1\colon \circ \qquad \Sigma \vdash t_2\colon \circ}{\Sigma \vdash t_1 \vee t_2\colon \circ} \qquad \frac{\Sigma \vdash t\colon \circ}{\Sigma \vdash \neg t\colon \circ} \qquad \frac{\Sigma \vdash t\colon \circ}{\Sigma \vdash \langle a\rangle t\colon \circ}$$

$$\frac{}{\Sigma, x\colon \tau \vdash x\colon \tau} \qquad \frac{\Sigma \vdash t_1\colon \tau_1 \to \tau_2 \qquad \Sigma \vdash t_2\colon \tau_1}{\Sigma \vdash (t_1\, t_2)\colon \tau_2} \qquad \frac{\Sigma, x\colon \tau_1 \vdash t\colon \tau_2}{\Sigma \vdash \lambda(x\colon \tau).\, t\colon \tau_1 \to \tau_2}$$

2.3.1 A Modal Lambda Calculus

We introduce a lambda calculus λ_{ML} over the set of operators of basic modal logic. Consider the set of *types* derived from the following grammar:

$$\tau ::= \circ \mid \tau \to \tau$$

Such a type system is called *simple* because it only has one type constructor, namely $\to$. The type $\circ$ is called the *base type*, all other types are *function types*. Types are assumed to be associative to the right, so any type can be written as $\tau_1 \to \cdots \to \tau_n \to \circ$. The order $\mathit{ord}(\tau)$ of a type is defined as $\mathit{ord}(\circ) = 0$, while the order of a function type $\tau_1 \to \tau_2$ is $\max\{\mathit{ord}(\tau_1) + 1, \mathit{ord}(\tau_2)\}$.

Fix a set of actions A, a set of propositions $\mathcal{P}$, and a set of *lambda variables* $\mathcal{V}$. Now let the following grammar define a set of *terms*

$$t ::= P \mid t \vee t \mid \neg t \mid \langle a\rangle t \mid x \mid (t\, t) \mid \lambda(x\colon \tau).\, t$$

where $x \in \mathcal{V}$ and τ is a type. A term of the form $\lambda(x\colon \tau).\, t$ is called *lambda abstraction*, and stands for an anonymous function that consumes an argument x of type τ and returns t, or rather its value considering the value of x. A term of the form $(t_1\, t_2)$ is called *application* and feeds the right subterm to the left, which necessarily must be of a function type (see below for typing rules).

A collection of typing hypotheses of the form $\Sigma = x_1\colon \tau_1, \ldots, x_n\colon \tau_n$ is called a *context*. A term is called *well-typed* if the statement $\Sigma \vdash t\colon \tau$ can be derived from the rules in Figure 2.1 for some context Σ. For example, from the context $\Sigma = x\colon \circ$, we can derive the judgment $\Sigma \vdash x \vee P\colon \circ$ by an application of the axiom for propositions, and then a use of the rule for disjunctions. Given a well-typed term of λ_{ML}, the notions of *syntax tree*, *subterm*, and *free variable* are defined in the usual way. For such a well-typed term t, any subterm is naturally associated with a type as well. We refer to this type as the type of the subterm. Note that this type is not unique and depends on the exact derivation that witnesses that t is well-typed. An expression of the form $\lambda x.\, t$ binds all occurrences of the variable x in t. We tacitly assume that a variable occurs only with single type in a well-typed term.

A term of type $\circ$ then represents a formula of basic modal logic[6], while, for example, a term of type $\circ \to \circ$ is a function that consumes a formula of basic modal logic and returns a formula of basic modal logic. Obviously, any formula of basic modal logic has type $\circ$, and so does, for example,

$$\langle a\rangle \big((\lambda(x\colon \circ).\, \langle a\rangle x)\, \neg P\big).$$

[6]If the term has free variables, this requires some form of variable assignment.

On the other hand, the subterm $\lambda(x\colon \circ).\, \langle a\rangle x$ encodes a function that consumes a formula φ of basic modal logic and returns $\langle a\rangle\varphi$.

The last two examples are better understood in light of the three reduction rules coming with the Lambda Calculus in general, and hence, also λ_{ML}.

- The rule of *α-reduction*, also called α-conversion, allows us to rename variable binders and their bound variables by replacing a term of the form $\lambda(x\colon \tau).\, t$ by $\lambda(y\colon \tau).\, t[y/x]$. This works similarly to renaming of variables in e.g., first-order logic, and mostly serves the same purpose, namely to avoid *variable capture* in the context of β-reduction (see below) or another kind of substitution. For example, consider
$$(\lambda(y\colon \tau).\, x\, y)[y/x].$$
If this substitution is done straightforwardly, we obtain the term $\lambda(y\colon \tau).\, y\, y$, i.e., the free variable y of the substitution target is *captured* by the lambda abstraction. In order to avoid this, using α-conversion we can replace the original term by
$$(\lambda(z\colon \tau).\, x\, z)\, [y/x],$$
which yields the desired substituted term $\lambda(z\colon \tau).\, y\, z$. Substitution that does not induce variable capture is called *capture-avoiding.*

- The defining reduction rule of the Lambda Calculus is *β-reduction*, which allows us to replace a (sub)term of the form $t = (\lambda(x\colon \tau).\, t_1)\, t_2$ by $t' = t_1[t_2/x]$, where substitution is assumed to be capture-avoiding, but otherwise is defined as usual. In this case, we say that t *β-reduces* to t' and similarly for a term where t is a subterm. For example, the term $\langle a\rangle\big((\lambda x\colon\ \circ\ .\, \langle a\rangle x)\, \neg P\big)$ does β-reduce to $\langle a\rangle\langle a\rangle\neg P$, whence it is clear that both the term and its β-reduct represent a formula of basic modal logic. Note that, similarly to scoping rules in many logics, substitution in β-reduction does not work past lambda bindings of a variable of the same name, i.e., $(\lambda(x\colon \tau).\, \lambda(x\colon \tau).\, x)P$ does β-reduce to $\lambda(x\colon \tau).\, x$ and not to $\lambda(x\colon \tau).\, P$.

- The final reduction is η-reduction, which allows us to replace a term of the form $\lambda(x\colon \tau).\, f\, x$ by just f if x does not occur freely in f. This reduction is not central to the thesis and will not be used in the following. However, the inverse of this reduction, called η-expansion, will be used in Section 3.1.3 in order to help bringing HFL formulas into a special normal form.

Let t be a term in λ_{ML}. A *redex* (for reducible expression) is a subterm of the form $(\lambda(x\colon \tau)\, t_1)\, t_2$, i.e., a candidate for β-reduction. The subterm t_1 is the operator part of the redex, and t_2 is the operand of the redex. The order of a redex is the order of its left subterm.

We say that t *reduces* to a term t' if there is a sequence of terms $t = t_1, \ldots, t_n = t'$ such that, for all $1 \leq i \leq n-1$, we have that t_{i+1} can be obtained from t_i by one β-reduction or α-conversion. Such reductions are *confluent* in the sense that, if a term t reduces to t_1 and t_2 via two different chains of reductions, then there is t' such that both t_1 and t_2 reduce to t'. Moreover, reductions in λ_{ML} are *strongly normalizing* in the sense that, for every term t there is a term t' such that t reduces to t' and any possible sequence of reductions from t' consists entirely of α-conversions. Due to confluence, t' is unique up to α-conversion and, hence, can be considered the β *normal form* of t. We briefly sketch why such a term t' must exist following a proof pattern found e.g., in [9]. Due to confluence, it is enough to exhibit a reduction strategy that terminates, i.e., reaches a term such that no further β-reductions are possible after accounting for α-conversion.

Theorem 2.3.1. *Every well-typed term in a simply-typed lambda calculus is equivalent to one in β normal form.*

(Sketch). Let t be a term in λ_{ML}. Clearly, if t contains no redexes at all, it is in normal form.

Let n be the maximal order of a redex in t. Associate to t, and to all its reducts, a measure $m(t') = (i_n, \dots, i_1)$ where i_j is the number of redexes of order j in t'. A reduction from t to t' is *descending in m* if $m(t) = (i_n, \dots, i_1)$, $m(t') = (i'_n, \dots, i'_1)$ and there is k such that $i_k > i'_k$ and $i_{k'} = i'_{k'}$ for all $k' > k$. Intuitively, a reduction that is descending in k reduces the number of redexes of some order k and does not increase the number of redexes of order greater than k. It can, however, increase the number of redexes of lower order. Obviously, any chain of β-reductions that are descending in m, starting from t, must eventually reach a term t' such that $m(t') = (0, \dots, 0)$, which is the desired normal form.

Now let t' be t or some term obtained by using the following reduction strategy, and assume that it is not already in normal form. Let k be the highest order such that there are more than 0 redexes of order k. Let $t'' = (\lambda(x : \tau).\, t_1)\, t_2$ be a redex of order k such that t_2 does not contain any redexes of order k itself. Such a redex must exist for otherwise we obtain an infinite chain of strictly descending subterms, which contradicts our definition of terms of λ_{ML}. It is not hard to see that β-reducing t'' to $t_1[t_2/x]$ (after applying α-conversion if necessary) is decreasing in m: Clearly this reduction does not introduce new redexes of order higher than k, and the number of redexes of order k in $t_1[t_2/x]$ equals the number of redexes of that order in t_1, which is one less than the number of redexes of order k in t''. Following this reduction strategy eventually yields a term with no redexes at all, which is the desired normal form. $\square$

Note that the sequences of reductions necessary to transform a term into β normal form can be k-fold exponentially large for terms of order k [10]. We use a similar strategy to prove a property of the automaton model introduced in this thesis (see Section 4.2.6).

Two terms t_1 and t_2 are *β-equivalent* if they reduce to the same β normal form. Note that this equivalence in λ_{ML} is purely syntactic and not a priori related to semantic equivalence of formulas of basic modal logic, as λ_{ML} is a purely syntactic calculus and is oblivious to the semantics of the formulas it produces. However, it is not hard to see that β-equivalence for λ_{ML} entails semantic equivalence for terms that are formulas of basic modal logic. Also note that β-equivalence is decidable. It can be decided by, given two terms t_1 and t_2, applying β-reductions to both until the normal form is reached. The terms are equivalent if the two normal forms agree up to α-equivalence.

A term t is in *head normal form* if its topmost subterm is not a redex. Note that head normal form is not unique. For example, $\lambda(x : \circ).\, (\lambda(y : \circ).\, y\, P)$ and $\lambda(x : \circ).\, P$ are both in head normal form, and the former term β-reduces to the latter. Hence, a term in head normal form is not necessarily in β normal form.

2.3.2 Krivine's Abstract Machine

We now present a machine model for λ_{ML} that, upon input t, computes a head normal form of t. This machine works via *call-by-name*, i.e., given a β-reduction, the machine does not reduce the operand of the redex, but continues reducing the operator. This machines is called *Krivine's Abstract Machine* (KAM) and is due to Krivine [60]; the version presented here is a slight variant.

Closures and *environments* are defined via mutual recursion. An environment is either the empty root environment e^0 or it defines the value of a single[7] lambda variable as a closure. Moreover, it has a *parent environment* that can define further variables. A closure is a term together with an environment which defines the value of free variables of the term. Formally, a closure c and an environment e must be derivable from the grammar

$$\begin{aligned} e &::= e^0 \mid (x \mapsto c, e) \\ c &::= (t, e) \end{aligned}$$

where x is a lambda variable and t is a term of λ_{ML}. Variable lookup for environments is defined as follows:

$$\mathsf{lookup}(x, e) = \begin{cases} c & \text{if } e = (x \mapsto c, e') \\ \mathsf{lookup}(x, e') & \text{if } e = (y \mapsto c, e'), y \neq x \\ \text{undefined} & \text{else.} \end{cases}$$

A *configuration* of the KAM is a tuple of the form (c, Γ) where c is a closure (t, e) such that $(t \colon \tau_1 \to \cdots \to \tau_n \to \circ)$ holds, and the argument stack Γ is a stack of closures with contents $c_1, \ldots, c_{n'}$ from top to bottom, where $n' \leq n$ and $c_i = (t_i, e_i)$ such that $(t_i \colon \tau_i)$ holds for $1 \leq i \leq n'$. The starting configuration is $((t, e^0), \varepsilon)$.

From a given configuration $((t, e), \Gamma)$, the machine continues depending on the form of t:

- If t is of the form $t_1\, t_2$, then the next configuration is $((t_1, e), \Gamma')$ where Γ' is Γ with (t_2, e) pushed on top.
- If t is of the form $\lambda(x \colon \tau).\, t'$ and Γ is nonempty, then the next configuration is $((t', e'), \Gamma')$ where $e' = (x \mapsto c, e)$ and Γ' is Γ without its top element, which is c.
- If t is of the form x, then the next configuration is $(\mathsf{lookup}(x, e), \Gamma)$.
- Otherwise, the machine halts.

Note that the machine has no support for the operators of modal logic. If the top operator of the input term is such an operator, the machine halts.

Example 2.3.2. Let $(\lambda(x \colon \circ).\, x\, P)\, (\lambda(z \colon \circ).\, \langle a \rangle y$ be a well-typed term under the context $\Sigma = y \colon \circ$. The Krivine Machine computes a head normal form for this term

[7]Whether an environment binds a single variable or several varies from implementation to implementation. In fact, we are going to use a multi-variable version later in this thesis, but restrict ourselves to a single-variable variant here for the sake of clarity.

as follows. We write down configurations on the left, and environments on the right.

$$\begin{array}{rl}
((\lambda(x\colon \circ).\, x\, P)\, (\lambda(z\colon \circ).\, \langle a\rangle y, e^0, \varepsilon) & \\
((\lambda(x\colon \circ).\, x\, P), e^0, (\lambda(z\colon \circ).\, \langle a\rangle y, e^0)) & \\
((x\, P, e_1), \varepsilon) & e_1 = (x \mapsto (\lambda(z\colon \circ).\, \langle a\rangle y, e^0), e^0) \\
((x, e_1), (P, e_1) & \\
((\lambda(z\colon \circ).\, \langle a\rangle y, e^0), (P, e_1)) & \\
((\langle a\rangle y, e_2), \varepsilon) & e_2 = (z \mapsto (P, e_1))
\end{array}$$

Since $\langle a\rangle(y \vee z)$ is not a redex, the machine halts.

2.4 Higher-Order Modal Fixpoint Logic

2.4.1 Simple Types for HFL

The set Types of simple types for HFL is defined inductively via

$$\tau ::= \bullet \mid \tau^v \to \tau$$

where $\bullet$ is called the ground type, a type of the form $\tau_1^v \to \tau_2$ is called a *function type*, and $v \in \mathcal{V} = \{+, -, \pm\}$ is called a *variance*. A variance indicates whether the argument decorated by it is to be used in a monotonic, antitonic or unrestricted fashion. The operator $\to$ associates to the right, hence, every type can be written in the form $\tau_1^{v_1} \to \cdots \to \tau_n^{v_n} \to \bullet$. We define the *maximal arity* ma of a type via

$$\mathsf{ma}(\bullet) = 0$$
$$\mathsf{ma}(\tau_1^{v_1} \to \cdots \to \tau_n^{v_n} \to \bullet) = \max(n, \mathsf{ma}(\tau_1^{v_1}), \ldots, \mathsf{ma}(\tau_n^{v_n}))$$

The maximal *order ord* of a type is defined via

$$\mathit{ord}(\bullet) = 0$$
$$\mathit{ord}(\tau_1^{v_1} \to \cdots \to \tau_n^{v_n} \to \bullet) = \max(\mathit{ord}(\tau_1) + 1, \ldots, \mathit{ord}(\tau_n) + 1)$$

The semantics of types are defined with respect to a given LTS and are complete lattices. The semantics of the ground type over a given LTS $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ is the partially ordered set

$$[\![\bullet]\!]_{\mathcal{T}} = (2^S, \sqsubseteq_\bullet)$$

where $\sqsubseteq_\bullet$ is ordinary set inclusion $\subseteq$. We know from Example 2.1.2 that this is a complete lattice. We also know from Example 2.1.2 that the set of functions into a complete lattice is a complete lattice itself if ordered pointwise. Given a lattice $(S, \leq)$, let $(S, \leq)^v$ be defined as $(S, \leq)^+ = (S, \leq)$, $(S, \leq)^- = (S, \geq)$ and $(S, \leq)^\pm = (S, \leq \cap \geq)$. Then the semantics of the type $\tau_1^v \to \tau_2$ is the inductively defined pair

$$[\![\tau_1^v \to \tau_2]\!]_{\mathcal{T}} = \big(([\![\tau_1]\!]_{\mathcal{T}})^v \to [\![\tau_2]\!]_{\mathcal{T}}, \sqsubseteq_{\tau_1^v \to \tau_2}\big)$$

where $([\![\tau_1]\!]_{\mathcal{T}})^v \to [\![\tau_2]\!]_{\mathcal{T}}$ is the set of monotonic functions from $[\![\tau_1]\!]_{\mathcal{T}}$ to $[\![\tau_2]\!]_{\mathcal{T}}$ if $v = +$, the set of antitonic functions from $[\![\tau_1]\!]_{\mathcal{T}}$ to $[\![\tau_2]\!]_{\mathcal{T}}$ if $v = -$, and the set of all functions from $[\![\tau_1]\!]_{\mathcal{T}}$ to $[\![\tau_2]\!]_{\mathcal{T}}$ if $v = \pm$. Moreover, the order $\sqsubseteq_{\tau_1^v \to \tau_2}$ is obtained by ordering the functions from $[\![\tau_1]\!]_{\mathcal{T}}^v$ to $[\![\tau_2]\!]_{\mathcal{T}}$ pointwise.

For a type τ of order k, the height $\mathsf{ht}([\![\tau]\!]_{\mathcal{T}})$ is k-fold exponential in the size of $\mathcal{T}$. More precise estimates can be found in [6].

2.4.2 The Syntax of HFL

Fix a set of actions A and a set of propositions $\mathcal{P}$. Moreover, fix a set $\mathcal{X}$ of fixpoint variables and a set $\mathcal{F}$ of lambda variables. We use symbols $X, Y, \dots$ for variables from $\mathcal{X}$ and symbols $x, y, \dots$ for variables from $\mathcal{F}$.

The syntax of HFL is defined inductively. We say that φ is an HFL *preformula* if it can be derived from the following grammar:

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

where $P \in \mathcal{P}$, $a \in A$, $x \in \mathcal{F}$, $v \in \mathcal{V}$, $\tau \in \mathsf{Types}$, and $X \in \mathcal{X}$. The operators

$$\begin{aligned} \varphi \wedge \psi &\equiv \neg(\neg\varphi \vee \neg\psi) \\ \varphi \to \psi &\equiv \neg\varphi \vee \psi \\ \mathtt{tt} &\equiv P \vee \neg P \\ \mathtt{ff} &\equiv \neg\mathtt{tt} \\ [a]\varphi &\equiv \neg\langle a\rangle\neg\varphi, \end{aligned}$$

where P is arbitrary, are defined in the usual way. If the distinction is not important, we will often treat these similarly to the built-in operators defined above; we will make explicit whether the additional operators are to be considered syntactic sugar or not if the distinction becomes important. Note that the greatest-fixpoint operator cannot be obtained in this way so easily since negation is available only at ground type. We use σ to denote either of μ, ν. For example, $\sigma(X : \tau).\,\varphi$ stands for either of $\mu(X : \tau).\,\varphi$ and $\nu(X : \tau).\,\varphi$, where the *polarity* of σ is not important.

Well-typed Formulas

The above definition of the syntax of HFL allows us to write down preformulas that obviously cannot be endowed with a useful semantics, for example $(\langle a\rangle P\,\neg Q)$, or $(P \vee \lambda(x^+ : \bullet \to \bullet).\,Q)$. In order to avoid dealing with such objects, we employ a *type system* that filters out those preformulas that cannot be endowed with proper semantics. The remaining preformulas will be called HFL *formulas*.

A sequence Σ of the form $(X_1^{v_1} : \tau_1), \dots, (X_n^{v_n} : \tau_n), (x_1^{v'_1} : \tau'_1), \dots, (x_m^{v'_m} : \tau'_m)$ in which each fixpoint variable and each lambda variable occurs at most once is called a *context* and stores type assumptions for intermediate steps in the type derivation process. The empty context $\emptyset$ denotes a sequence with no assumptions, i.e., a sequence of length 0. Given a context Σ, the negated context Σ^- is obtained by reversing all variance decorations of variables from $+$ to $-$ and vice versa, on both fixpoint variables and lambda variables. Variance decorations with $\pm$ stay fixed, and decorations in the respective type also do not change.

Example 2.4.1. Consider the context

$$\Sigma = X_1^+ : \bullet \to \bullet, X_2^\pm : \bullet \to \bullet, x^- : (\bullet^+ \to \bullet)^- \to \bullet.$$

Its negated context is

$$\Sigma^- = X_1^- : \bullet \to \bullet, X_2^\pm : \bullet \to \bullet, x^+ : (\bullet^+ \to \bullet)^- \to \bullet.$$

Note that the decorations in, e.g., $(\bullet^+ \to \bullet)^- \to \bullet$ have not changed.

Figure 2.2: The HFL typing rules.

$$\frac{}{\Sigma \vdash P\colon \bullet} \qquad \frac{\Sigma \vdash \varphi\colon \bullet \qquad \Sigma \vdash \psi\colon \bullet}{\Sigma \vdash \varphi \lor \psi\colon \bullet} \qquad \frac{\Sigma^- \vdash \varphi\colon \bullet}{\Sigma \vdash \neg\varphi\colon \bullet} \qquad \frac{\Sigma \vdash \varphi\colon \bullet}{\Sigma \vdash \langle a\rangle\varphi\colon \bullet}$$

$$\frac{\Sigma \vdash \varphi\colon \tau_1^+ \to \tau_2 \qquad \Sigma \vdash \psi\colon \tau}{\Sigma \vdash \varphi\,\psi\colon \tau_2} \qquad \frac{\Sigma \vdash \varphi\colon \tau_1^- \to \tau_2 \qquad \Sigma^- \vdash \psi\colon \tau_1}{\Sigma \vdash (\varphi\,\psi)\colon \tau_2}$$

$$\frac{\Sigma \vdash \varphi\colon \tau_1^\pm \to \tau_2 \qquad \Sigma \vdash \psi\colon \tau_1 \qquad \Sigma^- \vdash \psi\colon \tau_1}{\Sigma \vdash \varphi\,\psi\colon \tau_2} \qquad \frac{}{\Sigma, x^+ \colon \tau \vdash x\colon \tau}$$

$$\frac{\Sigma, x^v \colon \tau_1 \vdash \varphi\colon \tau_2}{\Sigma \vdash \lambda(x^v\colon \tau_1).\,\varphi\colon \tau_1^v \to \tau_2} \qquad \frac{}{\Sigma, X^+ : \tau \vdash X : \tau} \qquad \frac{\Sigma, (X^+ : \tau_1) \vdash \varphi\colon \tau_1}{\Sigma \vdash \mu(X : \tau_1).\,\varphi\colon \tau_1}$$

$$\frac{\Sigma, (X^+ : \tau_1) \vdash \varphi\colon \tau_1}{\Sigma \vdash \nu(X : \tau_1).\,\varphi\colon \tau_1}$$

We say that an HFL preformula φ has type τ in context Σ if $\Sigma \vdash \varphi\colon \tau$ can be derived via the rules in Figure 2.2. In this case, we say that φ is *well-typed.* We are particularly interested in well-typed closed (see below) formulas of ground type, i.e., formulas φ such that $\emptyset \vdash \varphi\colon \bullet$ can be derived. Type derivations are unique [90], which justifies to speak of *the* type of a given formula. Since such a type derivation associates a type with each subformula, this also allows us to speak of the type of a subformula (to be defined in the next section) of a well-typed formula.

Since typing annotations are often not needed in full detail, or not at all, we only display typing information when necessary. Moreover, we usually suppress the variance annotations on the type, if they are not needed. In both cases, lambda variables and fixpoint variables are tacitly assumed to be annotated such that the formula in question is well-typed.

Syntactic Conventions

Definition 2.4.2. Given an HFL formula φ, we define the set of its subformulas $\mathrm{sub}(\varphi)$ inductively via

$$\begin{aligned}
\mathrm{sub}(\mathsf{P}) &= \{P\}\\
\mathrm{sub}(\psi_1 \lor \psi) &= \{\psi_1 \lor \psi_2\} \cup \mathrm{sub}(\psi_1) \cup \mathrm{sub}(\psi_2)\\
\mathrm{sub}(\neg\psi) &= \{\neg\psi\} \cup \mathrm{sub}(\psi)\\
\mathrm{sub}(\langle\mathsf{a}\rangle\psi) &= \{\langle a\rangle\psi\} \cup \mathrm{sub}(\psi)\\
\mathrm{sub}(\psi_1\,\psi_2) &= \{\psi_1\,\psi_2\} \cup \mathrm{sub}(\psi_1) \cup \mathrm{sub}(\psi_2)\\
\mathrm{sub}(\mathsf{x}) &= \{x\}\\
\mathrm{sub}(\lambda(\mathsf{x}^{\mathsf{v}}\colon \tau).\,\psi) &= \{\lambda\tau.\,\varphi\} \cup \mathrm{sub}(\psi)\\
\mathrm{sub}(\mathsf{X}) &= \{X\}\\
\mathrm{sub}(\mu(\mathsf{X}\colon \tau).\,\psi) &= \{\mu X.\,\varphi\} \cup \mathrm{sub}(\psi).
\end{aligned}$$

Definition 2.4.3. Given an HFL formula φ, we define its *formula tree* as a ranked tree with labels in $\mathrm{sub}(\varphi)$ inductively as follows. The root is labeled by φ, and the labels of the sons of a node depend on the label of the node itself:

- Nodes labeled by P, x or X are leaves.
- A node t labeled by $\langle a\rangle\psi, \neg\psi, \lambda x.\, \psi$ or $\mu X.\, \psi$ has one successor $t0$ labeled by ψ.
- A node labeled by $\psi_1 \vee \psi_2$ or $\psi_1\, \psi_2$ has two successors $t0$ labeled by ψ_1 and $t1$ labeled by ψ_2.

The formula DAG of φ is then obtained by identifying isomorphic subtrees as per usual. Note that necessarily the formula DAG of φ has $|\mathrm{sub}(\varphi)|$ many nodes. Hence we define the *size* of an HFL-formula φ as $|\mathrm{sub}(\varphi)|$, i.e., the size of its formula DAG.

Given an HFL formula φ, an *occurrence* of a subformula, in particular of a lambda variable of a fixpoint variable, is a node in the formula tree or formula DAG of φ that is labeled by that subformula. Whether we consider occurrences with respect to trees or DAGs is clear from context or stated explicitly in this thesis, provided that the difference is actually meaningful.

An occurrence of a fixpoint variable X or a lambda variable x is *bound* at the first formula $\sigma X.\, \psi$, respectively $\lambda x.\, \psi$ in the syntax tree or DAG above it, provided that the binding formula matches variable name and type. If there is no such binding formula, the variable occurs *freely.*

Definition 2.4.4. The set of *free* variables $\mathsf{free}(\psi) \subseteq \mathcal{X} \cup \mathcal{F}$ is defined inductively for each subformula ψ of some HFL-formula φ.

$$\begin{aligned}
\mathsf{free}(P) &= \emptyset \\
\mathsf{free}(\varphi \vee \psi) &= \mathsf{free}(\varphi) \cup \mathsf{free}(\psi) \\
\mathsf{free}(\neg\varphi) &= \mathsf{free}(\varphi) \\
\mathsf{free}(\langle a\rangle\varphi) &= \mathsf{free}(\varphi) \\
\mathsf{free}((\varphi\, \psi)) &= \mathsf{free}(\varphi) \cup \mathsf{free}(\psi) \\
\mathsf{free}(x) &= \{x\} \\
\mathsf{free}(\lambda((x^v : \tau)).\, \varphi) &= \mathsf{free}(\varphi) \setminus \{x\} \\
\mathsf{free}(X) &= \{X\} \\
\mathsf{free}(\mu((X : \tau)).\, \varphi) &= \mathsf{free}(\varphi) \setminus \{X\}
\end{aligned}$$

Note that, for each free variable of a formula, there is a free occurrence of that variable in the formula. A formula or one of its subformulas is said to be *closed* if it has no free variables. It is fixpoint closed, respectively lambda-variable closed if it has no free variables of the respective kind.

We call an HFL formula φ *well-named* if it is closed and

- for each fixpoint variable X, the formula φ contains exactly one subformula of the type $\sigma(X : \tau).\, \psi$, and

- for each lambda variable x, the formula φ contains exactly one subformula $\lambda(x^v \colon \tau).\, \psi$.

For a well-named formula, similarly to the situation in $\mathcal{L}_\mu$, there is a partial function $\mathsf{fp}_\varphi \colon \mathcal{X} \to \mathrm{sub}(\varphi)$ that maps each fixpoint variable X that occurs in φ to the *defining formula* of X, defined as φ' in the unique subformula of the form $\sigma X.\, \varphi'$ of φ. Moreover, in a well-named formula φ, we can partially order the set of fixpoint variables that occur in it. Define $Y \succ X$ if and only if $\mathsf{fp}_\varphi(X)$ is a proper subformula of $\mathsf{fp}_\varphi(Y)$. In this case, we say that Y is *outermore* than X. Moreover, for each fixpoint variable X that is not maximal with respect to $\succ$, there is a unique fixpoint variable Y such that $Y \succ X$ and no other fixpoint variable Z exists with $Y \succ Z$ and $Z \succ X$. In this case, we set $Y = \mathit{upVar}(X)$. Note that this definition generalizes the definition (cf. Section 2.2.3) of $\succ$ for $\mathcal{L}_\mu$. The definition of $\succ$ of an $\mathcal{L}_\mu$ formula, considered as an HFL formula, coincides with the definition here.

Note the following difference between $\mathcal{L}_\mu$ and HFL here: In $\mathcal{L}_\mu$, $Y \succ X$ is equivalent to the fact that the semantics of X depends possibly on those of Y, and the semantics of Y depends on those of X. If two fixpoint variables are not comparable via $\succ$, then their semantics can only depend on each other in the sense that both depend on the semantics of a third fixpoint variable that is comparable to both of them. On the other hand, in HFL, dependencies are harder grasp. While $Y \succ X$ implies that the semantics of the two variables possibly depend on each other, the converse does not hold. Consider, for example

$$\big(\lambda(x \colon \bullet \to \bullet).\, \mu(X \colon \bullet).\, x\, X\big)\, \nu(Y \colon \bullet \to \bullet).\, \lambda(x \colon \bullet).\, Y\, x \vee x$$

in which X and Y are incomparable with respect to $\succ$. However, this formula β-reduces[8] to

$$\mu(X \colon \bullet).\, \big(\nu(Y \colon \bullet \to \bullet).\, \lambda(x \colon \bullet).\, Y\, x \vee x\big)\, X$$

where the dependency is much easier to see.

We say that a well-typed HFL formula $\emptyset \vdash (\varphi \colon \tau)$ has type order k if the highest type that occurs in the derivation of this judgment has order k, i.e., if k is the highest order of any subformula of φ. We denote the fragment of all HFL formulas of order at most k by HFL^k. Note that $\mathrm{HFL}^0 = \mathcal{L}_\mu$.

Substitution of the form $\varphi[\psi_1/\psi_2]$ works slightly differently depending on whether ψ_2 is either a fixpoint or lambda variable, or not. In the case of a lambda or fixpoint variable, substitution is assumed to replace all occurrences of ψ_2 that are free by ψ_1, while in the second case, all occurrences are substituted. For example, $(\mu X.\, X)[Y/X] = \mu X.\, X$, while

$$(\mu X.\, X \vee P)[(\neg X \wedge \neg P)/(X \vee P)] = \mu X. \neg X \wedge \neg P.$$

Which kind of substitution is meant to occur will be clear from context if not explicitly stated. Finally, in order to reduce notational clutter, we introduce notation for mass substitution. Let φ be an HFL formula, let I be a finite set and let $\{\psi_i \mid i \in I\}$ be a collection of subformulas of φ that are pairwise not subformulas of each other. Let $\{\psi_i' \mid i \in I\}$ be a set of HFL formulas such that, for all i, the formula ψ_i' has the same type as ψ_i under the typing hypothesis used to type ψ_i in φ. Moreover, assume that the ψ_i' are pairwise not subformulas of each other. Then

$$\varphi[\psi_i'/\psi_i \mid i \in I]$$

[8]The semantics of HFL is invariant under β-reduction, see Lemma 2.4.7 below.

Figure 2.3: The semantics of HFL formulas.

$$\begin{aligned}
[\![\Sigma \vdash P \colon \bullet]\!]^{\eta}_{\mathcal{T}} &= \{v \in [\![\bullet]\!]_{\mathcal{T}} \mid P \in \mathcal{L}(v)\} \\
[\![\Sigma \vdash \varphi \vee \psi \colon \bullet]\!]^{\eta}_{\mathcal{T}} &= [\![\Sigma \vdash \varphi \colon \bullet]\!]^{\eta}_{\mathcal{T}} \cup [\![\Sigma \vdash \psi \colon \bullet]\!]^{\eta}_{\mathcal{T}} \\
[\![\Sigma \vdash \neg\varphi \colon \bullet]\!]^{\eta}_{\mathcal{T}} &= [\![\bullet]\!]_{\mathcal{T}} \setminus [\![\Sigma \vdash \varphi \colon \bullet]\!]^{\eta}_{\mathcal{T}} \\
[\![\Sigma \vdash \langle a\rangle\varphi \colon \bullet]\!]^{\eta}_{\mathcal{T}} &= \{v \in [\![\bullet]\!]_{\mathcal{T}} \mid \text{ ex. } w \in [\![\bullet]\!]_{\mathcal{T}} \\
&\qquad\qquad \text{s.t. } w \in [\![\Sigma \vdash \varphi \colon \bullet]\!]^{\eta}_{\mathcal{T}} \text{ and } v \xrightarrow{a} w\} \\
[\![\Sigma \vdash (\varphi \colon\ \psi)\tau_1]\!]^{\eta}_{\mathcal{T}} &= [\![\Sigma \vdash \varphi \colon \tau_2^v \to \tau_1]\!]^{\eta}_{\mathcal{T}}([\![\Sigma \vdash \psi \colon \tau_2]\!]^{\eta}_{\mathcal{T}}) \\
[\![\Sigma \vdash x \colon \tau]\!]^{\eta}_{\mathcal{T}} &= \eta(x) \\
[\![\Sigma \vdash \lambda(x^v \colon \tau_1).\, \varphi \colon \tau_1^v \to \tau_2]\!]^{\eta}_{\mathcal{T}} &= f \in [\![\tau_1^v \to \tau_2]\!]_{\mathcal{T}} : \text{f.a. } y \in [\![\tau_1]\!]. \\
&\qquad\qquad f(y) = [\![\Sigma, y^v \colon \tau_1 \vdash \varphi \colon \tau_2]\!]^{\eta[x \mapsto y]}_{\mathcal{T}} \\
[\![\Sigma \vdash X \colon \tau]\!]^{\eta}_{\mathcal{T}} &= \eta(X) \\
[\![\Sigma \vdash \mu(X \colon \tau).\, \varphi \colon \tau]\!]^{\eta}_{\mathcal{T}} &= \bigsqcap\{d \in [\![\tau]\!]_{\mathcal{T}} \mid [\![\Sigma, X \colon \tau^{+} \vdash \varphi \colon \tau]\!]^{\eta[X \mapsto d]}_{\mathcal{T}} \sqsubseteq_{\tau} d\} \\
[\![\Sigma \vdash \nu(X \colon \tau).\, \varphi \colon \tau]\!]^{\eta}_{\mathcal{T}} &= \bigsqcup\{d \in [\![\tau]\!]_{\mathcal{T}} \mid d \sqsubseteq_{\tau} [\![\Sigma, X \colon \tau^{+} \vdash \varphi \colon \tau]\!]^{\eta[X \mapsto d]}_{\mathcal{T}}\}
\end{aligned}$$

denotes

$$(\cdots(\varphi[\psi'_{i_1}/\psi_{i_1}])\cdots)[\psi'_{i_n}/\psi_{i_n}]$$

where $i_1, \dots, i_n$ is any enumeration of I. Note that the conditions on the ψ_i and ψ'_i are such that this yields the same formula independently of the enumeration.

Example 2.4.5. Let $\mathcal{X} = \{X, Y\}$ and let

$$\varphi = \mu X.\,(X \vee P) \wedge \mu Y.\, Y \vee P$$

Let $\psi_X = X \vee P$ and let $\psi_Y = Y \vee P$. Let $\psi'_X = X$ and let $\psi'_Y = Y$. Then

$$\varphi[\psi'_Z/\psi_Z \mid Z \in \mathcal{X}] = \mu X.X \wedge \mu Y.\, Y.$$

The main purpose of this notation is to replace a set of variables indexed by a collection of fixpoint or lambda variables, as shown in the example above.

2.4.3 Semantics of HFL

In a similar manner to what we stipulated for $\mathcal{L}_\mu$, when discussing the semantics of an HFL formula over an LTS, we tacitly assume that the sets of actions and propositions involved match.

Consider a context Σ and an LTS $\mathcal{T}$. An *interpretation* respecting this context is a partial map η from $\mathcal{X} \cup \mathcal{F}$ into $\bigcup_{\tau \in \mathsf{Types}} [\![\tau]\!]_{\mathcal{T}}$ such that if $X^v \colon \tau$ appears in Σ, then $\eta(X) \in [\![\tau]\!]_{\mathcal{T}}$, and such that if $x \colon \tau'$ appears in Σ, then $\eta(x) \in [\![\tau']\!]_{\mathcal{T}}$. Note that we also allow interpretations to yield values for fixpoint variables, which is useful e.g., in the context of formulas with free fixpoint variables. The *update* $\eta[X \mapsto f]$

of an interpretation is defined as

$$\begin{aligned}\eta[X \mapsto f](Y) &= \eta(Y) \text{ if } Y \neq X\\ \eta[X \mapsto f](Y) &= f \text{ if } Y = X\\ \eta[X \mapsto f](x) &= \eta(x)\end{aligned}$$

and similarly for lambda variables.

Let φ be a well-typed HFL formula, i.e., one such that $\Sigma \vdash \varphi\colon \tau$ is derivable. The semantics $[\![\varphi]\!]^{\eta}_{\mathcal{T}}$ of φ over an LTS $\mathcal{T}$, and with respect to an interpretation η that respects Σ is defined inductively as in Figure 2.3. For a closed ground-type formula φ and a pointed LTS $\mathcal{T}, v$, we write $\mathcal{T}, v \models \varphi$ to denote that $v \in [\![\varphi]\!]_{\mathcal{T}}$. In this case, $\mathcal{T}, v$ is called a *model* of φ. We say that two well-typed HFL formulas φ and ψ are *equivalent* if they have the same type and, for all LTS $\mathcal{T}$ and for all interpretations η, we have that $[\![\varphi]\!]^{\eta}_{\mathcal{T}} = [\![\psi]\!]^{\eta}_{\mathcal{T}}$. In this case, we write $\varphi \equiv \psi$.

Example 2.4.6. Consider the HFL formula φ defined as

$$\Big(\mu(F\colon \tau \to \bullet).\, \lambda(g\colon \tau).\, g\, P \vee \big(F\, \lambda(y\colon \bullet).\, g\,(g\, y)\big)\Big)(\lambda(z\colon \bullet).\, \langle a\rangle z)$$

where $\tau = \bullet \to \bullet$. It says that $\langle a\rangle^{2^i} P$ holds for some $i \in \mathbb{N}$.

It is immediate from the definition of the semantics of HFL that it is invariant under β-reduction in the following sense.

Lemma 2.4.7. *Let $(\lambda(x\colon \tau).\, \varphi)\, \psi$ be a well-typed formula such that ψ is of type τ. Then $(\lambda(x\colon \tau).\, \varphi)\, \psi \equiv \varphi[\psi/x]$.*

Proof. By the semantics of HFL. □

Note that β-reduction maintains equivalence only if it appears at a redex in the sense of the lambda calculus. Generally,

$$\big(\mu(X : \tau).\, \lambda(x\colon \tau).\, \varphi\big)\, \psi \not\equiv \mu(X : \tau).\, \varphi[\psi/x].$$

This equality does not hold due to two reasons: The first reason is that $\lambda(x\colon \tau).\, \varphi$ has a different type than $\varphi[\psi/x]$, so the type τ of X on the right is wrong. The second reason is that, on the left side, the argument x is not fixed, i.e., recursive calls of X can be made to the value of X at a different argument, while on the right side, the value of what was x is fixed to be ψ. Clearly, this is not the same in general.

Moreover, the semantics of HFL is invariant under renaming of variables, i.e., the semantics of an HFL formula is invariant under replacing a bound variable and all its bound occurrences by a fresh one. Also, HFL is invariant under fixpoint unfolding. Given a fixpoint definition $\sigma(X : \tau).\, \varphi$, we have that

$$\sigma(X : \tau).\, \varphi \equiv \big(\varphi[\sigma(X : \tau).\ \varphi/X]\big).$$

This is called the *fixpoint unfolding principle*; its validity follows directly from the definition of the semantics of $\sigma(X : \tau).\, \varphi$ as a fixpoint.

Moreover, HFL is bisimulation invariant, i.e., closed ground-type HFL formulas cannot distinguish bisimilar vertices and can only define unions of bisimulation equivalence classes [90].

Finally, the satisfiability problem for HFL is undecidable, i.e., given a closed ground-type HFL formula, it is generally not possible to decide if it has a model. This is a property inherited [90] from FLC [70], a fragment of HFL[1].

2.4.4 HFL Model Checking

Already in [90], Viswanathan and Viswanathan observed that the global bottom-up model-checking algorithm [36] for $\mathcal{L}_\mu$ (see Section 2.2.3 for a sketch) can be extended to HFL. Given a finite LTS and, if necessary an interpretation, in $\mathcal{L}_\mu$, each subformula of a given formula defines a set, and the sets for non-leaf subformulas in the syntax tree can be obtained from the sets defined by those below them via the semantics of $\mathcal{L}_\mu$. In HFL, each formula defines either a function in some function type, or a set. Instead of computing the set represented by a formula, one can now compute the function represented by a formula by enumerating all possible arguments and evaluating the function on them. The semantics of fixpoint formulas can again be computed using the characterization via the Kleene Fixpoint Theorem. In this case, instead of initializing the value of the respective fixpoint variable by the full or empty set, the variable is initialized by the respective top or bottom element of the relevant type lattice, i.e., a function of the form $f\colon x_1, \dots, x_k \mapsto \top$, respectively $f\colon x_1, \dots, x_k \mapsto \bot$.

An analysis of the size of the respective function tables yields that, for a given type order k, a function of this type order has k-fold exponentially many potential arguments, yielding a table of the same size. Moreover, the height of the respective type lattices is in the same order. Together, this yields a k-EXPTIME upper bound for the complexity to compute the semantics of an HFL formula of order k. In [6], this complexity is analyzed in more detail, and a matching lower bound is established. In fact, under modest assumptions on the arities of the types involved, this bound already holds for *data complexity*, i.e, for a fixed formula.

It should be noted that the k-EXPTIME upper bound only holds for fixed k, if k is part of the input, an additional exponent in the size of the input formula is to be expected [6]. For HFL^1 and HFL^2, algorithms exploiting *neededness analysis* are available [5, 79] and promise considerable speedup in practice.

The model-checking procedure presented in [6] to re-establish upper bounds for HFL model-checking employs a (local) model-checking game. Over a given finite LTS, all fixpoint definitions are equivalent to a finite unraveling of length depending on their type. After converting the input formula into such an unraveling, the resulting formula is fixpoint-free. The authors of [6] then convert the model-checking problem into an alternating reachability game by extending the standard handling of boolean and modal operators by a procedure where, upon reaching a formula that is a function application, $\mathcal{V}$ can propose semantics for the operand side formula. $\mathcal{S}$ can either accept these semantics, in which case the game continues with an evaluation of the operator side formula with the semantics proposed by $\mathcal{V}$ bound to a given lambda variable, if necessary, or $\mathcal{S}$ does not accept the semantics proposed by $\mathcal{V}$, and the game then continues within the operand where $\mathcal{V}$ now has to show correctness of her semantics.

This construction has two drawbacks: First, an explicit enumeration of complete type spaces of the formulas in questions seems alien to the notion of a local model-checking game, already for the reason that such an enumeration potentially includes many functions that are not even HFL definable. As outlined in [6], this construction also does not mesh correctly with fixpoint definitions and strictly requires the elimination by unraveling of these beforehand. The model-checking game proposed in Section 3.2 does not have these problems, but, on the other hand, does not give rise to a competitive model-checking algorithm (cf. Section 3.3).

2.4.5 Acceptance Conditions for Higher-Order Logics

We close our discussion of HFL with some remarks on acceptance conditions for a prospective automaton model for HFL. Already from [6] and [62] it is clear that a standard parity condition cannot be enough. The authors of [6] report problems when trying to endow their model-checking game with a parity condition, and, hence resort to an unraveling technique to circumvent the problem. However, they argue that some kind of prioritization of the operator side of an application seems necessary to obtain correct semantics.

In [62], Lange gives a model-checking game for FLC, a fragment of HFL^1. Exploiting the LIFO-behavior of the semantics of the game in [62], a condition called *stack-increasing*, (nowadays called *stair parity condition* [66]) is enough to characterize occurrences of fixpoints that contribute to the acceptance condition and correctly separates occurrences that can be considered terminated subroutines. An HFL^1-version of a formula exhibited in this paper illustrates this behavior. Consider the formula

$$\Big(\mu(X : \bullet \to \bullet).\, \lambda(x : \bullet).\, x \vee \big(\nu(Y : \bullet).\, X\, Y\big)\Big)\, P$$

which is a condensed version of the example from [62]. Consider an intuitive model-checking game over some LTS that is not important. $\mathcal{V}$ chooses a subformula at a disjunction, and at applications, evaluation of the operator side continues first while we remember the value of the operand. If $\mathcal{V}$ does not choose the left disjunct in the initial part of the play, the game will now enter the fixpoint definition of Y, which resolves to $X\,Y$, which, in turn, resolves to the defining formula of Y again. If $\mathcal{V}$ now always chooses the left disjunct, which is just x and binds to Y again, we return to the defining formula of Y. The game now potentially loops in such a sequence of turns, where both X and Y appear infinitely often. However, a simple unfolding argument shows that the formula is equivalent to `tt`. By unfolding X in the definition of Y, we obtain an alternative definition as

$$\ldots \nu Y.\, \big(\lambda x \vee \ldots\big)\, Y \ldots$$

which then β-reduces to a formula containing $\nu Y.\, Y \vee \ldots$. Hence, a simple parity condition, which, if continuing the behavior of $\mathcal{L}_\mu$, would assign a higher precedence to X since it is outermore, does not properly reflect the semantics of this formula.

An equivalent formulation of the condition in [62] generates a binary tree out of a play of the game which branches on configurations with a *chop*-operator, a weak version of function application. The part of the play that corresponds to the operator part of the play continues on the left, and the part that corresponds to the operand part of the play, if it exists, continues on the right. It can then be shown that this tree contains exactly one infinite path, which can be obtained by always continuing in right, i.e., on the operand side, if possible, and that a configuration for a fixpoint variable is stack-increasing if and only if it is on this path. Hence, a simple parity condition on the configurations on the path correctly captures semantics. Since the play in the model-checking game for FLC always continues on the left side, this, again, yields a kind of prioritization of the operator side of an application, respectively a chop operator.

Unfortunately, HFL only shares the simple LIFO behavior of FLC at type order 1. See Section 6.2.1 for a discussion on the LIFO behavior in HFL^1 and, in particular, Examples 6.2.13 and 6.2.14 for why, already at type order 2, no LIFO behavior occurs. However, the notion of an unfolding tree is still useful in the context of HFL. The reason for this is that it isolates infinite recursion from higher-order effects by separating evaluations of the operator part of a function application from the operand. Similarly to the situation in FLC, an infinite path in an unfolding tree captures infinite recursion, and a parity condition on the configurations on the infinite path is enough to capture the semantics of HFL. However, in the context of HFL, it is much harder to establish the existence of such a unique path, and, unfortunately, it does not behave as predictably as it does in the context of FLC, i.e., by continuing always on the right. We devote Sections 6.2.1 and 6.2.2 to settings where it does.

Chapter 3

A Model-Checking Game for HFL

In this chapter, we develop a new model-checking game for HFL. The purpose of this game is to give alternative, operational semantics to HFL, since the model-checking game serves as an intermediate in the correctness proof of APKA, i.e., the proof that APKA are equi-expressive to HFL. This is useful since the rather complicated structure of how infinite recursion manifests itself in a run of an APKA is hard to synchronize with the denotational semantics of HFL directly.

A model-checking game for HFL exists [6], however, it does not fit the requirements, in particular full support for infinite structures, and guaranteed infinite plays if $\mathcal{V}$ wins. The latter gives rise to an asymmetric winning condition. Besides positions where one of the players has unequivocally won, i.e., at positions representing trivial questions such as whether a vertex satisfies a given proposition, $\mathcal{V}$ can lose the game by running out of space on a counter that regulates her remaining allowance of infinite recursion. On the other hand, $\mathcal{S}$ has no such limits, hence plays where $\mathcal{V}$ cannot force an outright win at a trivial position go on forever. This is by design, since it allows $\mathcal{V}$ to generate a winning strategy in the acceptance game of an associated APKA. Since plays of these games typically go on indefinitely, so should the model-checking game. The game can be found in Section 3.2, we briefly analyze its complexity in Section 3.3. It turns out that this model-checking game is clearly not a competitive algorithm, it exhibits a nonelementary blowup of the game graph already on very simple formulas. Rather, the game should be understood at what it was designed for: a semantic tool.

In order to make management of the game simpler, and since it is only needed on a special subclass of HFL formulas, the presentation of the game is preceded by a normal form called *automaton normal form* (ANF). This normal form mirrors the syntax of APKA (hence the name). In particular, it synchronizes lambda abstraction and fixpoint unfolding by allowing the former only to be used in conjunction with the latter. This makes the interaction of higher-order constructs and recursion much more explicit and disallows certain kinds of hidden parameters to fixpoints tucked away behind lambda abstraction. The exposition of ANF is done in Section 3.1.3.

As a preparation of the exposition of ANF, we re-develop a result of Lozes [67], namely that HFL admits negation normal form (NNF). Both the exact translation as well as the proof are new. In conjunction with this, we also present an alternative method to complement a formula by reversing the polarity of all operators. The advantage of this is that such a complementation does not break negation normal form, since negation symbols only appear in front of propositions. We develop the arguments around NNF in Section 3.1.2, the complementation procedure can be found in Section 3.1.1.

3.1 Normal Forms for HFL

3.1.1 A Simple Complementation Procedure

Given an HFL formula, it is straightforward to obtain a formula that expresses its negation without just negating it. This is done by reversing the polarity of all operators, e.g., switching $\vee$ with $\wedge$, and by interchanging least- and greatest-fixpoint operators. The advantage of this procedure is that it does not introduce negations except in front of propositions. Hence, this does not break negation normal form (see Section 3.1.2). We devote this section to show that this procedure actually correctly complements a given formula.

Definition 3.1.1. Let φ be an HFL formula with fixpoint variables in $\mathcal{X}$, lambda variables $\mathcal{F}$ and propositions $\mathcal{P}$. Let $\widetilde{\mathcal{X}} = \{\widetilde{X} \mid X \in \mathcal{X}\}$ and let $\widetilde{\mathcal{F}} = \{\widetilde{x} \mid x \in \mathcal{F}\}$. We define a *pseudo-complement* $\overline{\varphi}^p$ of φ by induction over the syntax of φ:

$$\begin{aligned}
\overline{P}^p &= \neg P \text{ if } P \in \mathcal{P}\\
\overline{\neg P}^p &= P \text{ if } P \in \mathcal{P}\\
\overline{\varphi_1 \vee \varphi_2}^p &= \overline{\varphi_1}^p \wedge \overline{\varphi_2}^p\\
\overline{\varphi_1 \wedge \varphi_2}^p &= \overline{\varphi_1}^p \vee \overline{\varphi_2}^p\\
\overline{\langle a\rangle\varphi'}^p &= [a]\overline{\varphi'}^p\\
\overline{[a]\varphi'}^p &= \langle a\rangle\overline{\varphi'}^p\\
\overline{\neg\varphi'}^p &= \neg\overline{\varphi'}^p \text{ if } \varphi' \notin \mathcal{P}\\
\overline{\varphi_1\,\varphi_2}^p &= \overline{\varphi_1}^p\,\overline{\varphi_2}^p\\
\overline{\lambda(x^v\colon \tau).\,\varphi'}^p &= \lambda((\widetilde{x})^v\colon \tau).\,\overline{\varphi'}^p\\
\overline{x}^p &= \widetilde{x}\\
\overline{\mu(X\colon \tau).\,\varphi'}^p &= \nu(\widetilde{X}\colon \tau).\,\overline{\varphi'}^p\\
\overline{\nu(X\colon \tau).\,\varphi'}^p &= \mu(\widetilde{X}\colon \tau).\,\overline{\varphi'}^p\\
\overline{X}^p &= \widetilde{X}
\end{aligned}$$

Observation 3.1.2. If φ is a well-typed HFL formula, then so is $\overline{\varphi}^p$, and both formulas have the same type and order. Moreover, $\overline{\varphi}^p$ has fixpoint variables in $\widetilde{\mathcal{X}}$ and lambda variables in $\widetilde{\mathcal{F}}$. Finally, $\varphi \equiv \overline{\overline{\varphi}^p}^p$, since the formulas are identical up to renaming of variables.

A proof for well-typedness for $\overline{\varphi}^p$ can be obtained from that for φ, since pseudo-complementation only reverses operators that naturally appear in pairs and behave similarly with respect to typing. We will make use of the last observation by tacitly identifying φ and $\overline{\overline{\varphi}^p}^p$ from now on.

Remark 3.1.3. Note that this pseudo-complement generally does not define the actual lattice complement. For function types, the latter can be seen to be the pointwise complement, i.e., for some function f and an argument x of a suitable type, we have that $\overline{f}(x) = \overline{f(x)}$. Consider, for example, the function defined by $\varphi = \lambda(x\colon \bullet).\, \neg x$. Over any given LTS, the complement of this function is defined by $\lambda(x\colon \bullet).\, x$: For any LTS $\mathcal{T}$, and any subset T of the underlying set S of $\mathcal{T}$, we have that

$$\begin{aligned}
&[\![\lambda(x\colon \bullet).\, \neg x]\!]_{\mathcal{T}}(T) \cup [\![\lambda(x\colon \bullet).\, x]\!]_{\mathcal{T}}(T)\\
=&(S \setminus T) \cup T\\
=&S
\end{aligned}$$

and

$$\begin{aligned}
&[\![\lambda(x\colon \bullet).\, \neg x]\!]_{\mathcal{T}}(T) \cap [\![\lambda(x\colon \bullet).\, x]\!]_{\mathcal{T}}(T)\\
=&(S \setminus T) \cap T\\
=&\emptyset,
\end{aligned}$$

which proves that the functions defined by $\lambda(x\colon \bullet).\, \neg x$ and $\lambda(x\colon \bullet).\, x$ are complements of each other.

On the other hand, $\overline{\lambda(x\colon \bullet).\, \neg x}^{p} = \lambda(\widetilde{x}\colon \bullet).\, \overline{\neg x}^{p} = \lambda(\widetilde{x}\colon \bullet).\, \neg\widetilde{x}$, which is equivalent to φ via renaming, and clearly defines a different function than $\lambda(x\colon \bullet).\, x$.

Let $\mathcal{T}$ be an LTS. Consider the following lattice pseudo-complement defined on the lattices associated to the HFL types via

$$\begin{aligned}
\overline{x}^{s} &= \overline{x} && \text{if } x \text{ is of ground type}\\
\overline{f}^{s} &= x \mapsto \overline{f(\overline{x}^{s})}^{s} && \text{if } f\colon x \mapsto f(x) \text{ is of a function type.}
\end{aligned}$$

Clearly, $\overline{\overline{x}^{s}}^{s} = x$ for ground type lattice elements. By an induction over the construction of the respective type, we also get that $f = \overline{\overline{f}^{s}}^{s}$ for function type lattice elements. Finally, by another induction over the construction of the respective types, we get that if f and g are lattice elements of type τ, and if $f \sqsubseteq_{\tau} g$, then $\overline{g}^{s} \sqsubseteq_{\tau} \overline{f}^{s}$. Hence, passing to the lattice pseudo-complement inverts the order of elements.

We now characterize the semantics of pseudo-complementation as defining the above lattice pseudo-complement. Note that, for each HFL type τ and each LTS $\mathcal{T}$, the lattice $[\![\tau]\!]_{\mathcal{T}}$ is always a boolean lattice due to Observation 2.1.3, and, hence, each element of $[\![\tau]\!]_{\mathcal{T}}$ has a lattice complement.

Lemma 3.1.4. *Let φ be an* HFL *formula and let $\mathcal{T}$ be an LTS and let η be an interpretation. Then $\overline{\varphi}^{p}$ defines a lattice pseudo-complement of the semantics of φ. Formally, we have that*

$$[\![\overline{\varphi}^{p}]\!]_{\mathcal{T}}^{\overline{\eta}^{p}} = \overline{[\![\varphi]\!]_{\mathcal{T}}^{\eta}}^{s}$$

where $\overline{\eta}^{p}$ is defined to yield

$$\begin{aligned}
\overline{\eta}^{p}(\widetilde{x}) &= \overline{\eta(x)}^{s}\\
\overline{\eta}^{p}(\widetilde{X}) &= \overline{\eta(X)}^{s}.
\end{aligned}$$

In particular, for a closed formula φ of ground type, we have that $\overline{\varphi}^{p} \equiv \neg\varphi$.

Proof. The last statement is a special case of the statement of the lemma, which we show by induction over the syntax tree of φ. Let $\mathcal{T}$ and η be an LTS and an interpretation.

- If φ is P or $\neg P$, the claim is immediate.
- If $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \neg\varphi'$, the claim is by simple boolean set manipulation.
- If $\varphi = \langle a \rangle \varphi$ or $\varphi = [a]\varphi$, the claim follows from the duality of the modal operators and the induction hypothesis.
- For $\varphi = X$ and $\varphi = x$, the claim follows from the definition of $\overline{\eta}^p$.
- If $\varphi = \varphi_1\,\varphi_2$, let φ_1 be of type $\tau_1 \to \tau_2$ and let $[\![\varphi_1]\!]^{\eta}_{\mathcal{T}} \in [\![\tau_1 \to \tau_2]\!]_{\mathcal{T}}$ equal $f \colon x \mapsto f(x)$. Moreover, let $[\![\varphi_2]\!]^{\eta}_{\mathcal{T}} \in [\![\tau_1]\!]_{\mathcal{T}}$ equal g. Then, by the induction hypothesis, $[\![\overline{\varphi_1}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = f' \colon x \mapsto \overline{f(\overline{x}^s)}^s$, and $[\![\overline{\varphi_2}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = \overline{g}^s$. But then

  $$\begin{aligned}[\![\overline{\varphi_1\,\varphi_2}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = [\![\overline{\varphi_1}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}}\,[\![\overline{\varphi_2}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} &= f'\,\overline{g}^s \\ &= \overline{f(\overline{\overline{g}^s}^s)}^s = \overline{f(g)}^s = \overline{[\![\varphi_1\,\varphi_2]\!]^{\eta}_{\mathcal{T}}}^s.\end{aligned}$$

- If $\varphi = \lambda x.\,\varphi'$ then, by the induction hypothesis, $[\![\overline{\varphi'}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = \overline{[\![\varphi']\!]^{\eta}_{\mathcal{T}}}^s$. Hence, $[\![\lambda x.\,\varphi']\!]^{\eta}_{\mathcal{T}} = f \colon y \mapsto [\![\varphi']\!]^{\eta[x \mapsto y]}_{\mathcal{T}}$, and

  $$\begin{aligned}\overline{[\![\lambda x.\,\varphi']\!]^{\eta}_{\mathcal{T}}}^s = f' \colon y \mapsto \overline{[\![\varphi']\!]^{\eta[x \mapsto \overline{y}^s]}_{\mathcal{T}}}^s& \\ = f' \colon y \mapsto [\![\overline{\varphi'}^p]\!]^{\overline{\eta}^p[\widetilde{x} \mapsto \overline{\overline{y}^s}^s]}_{\mathcal{T}} &= [\![\lambda \widetilde{x}.\,\overline{\psi'}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = [\![\overline{\varphi}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}}.\end{aligned}$$

- If $\varphi = \mu(X : \tau).\,\varphi'$, then

  $$f = [\![\mu X.\,\varphi']\!]^{\eta}_{\mathcal{T}} = \bigsqcap\{g \in [\![\tau]\!]_{\mathcal{T}} \mid [\![\varphi']\!]^{\eta[X \mapsto g]}_{\mathcal{T}} \sqsubseteq_\tau g\}.$$

  We show that

  $$\overline{f}^s = [\![\overline{\varphi}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = [\![\nu(\widetilde{X} : \tau).\,\overline{\varphi'}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}} = \bigsqcup\{g \in [\![\tau]\!]_{\mathcal{T}} \mid g \sqsubseteq_\tau [\![\overline{\varphi'}^p]\!]^{\overline{\eta}^p[\widetilde{X} \mapsto g]}_{\mathcal{T}}\}.$$

  By the induction hypothesis, we have $\overline{[\![\varphi']\!]^{\eta}_{\mathcal{T}}}^s = [\![\overline{\varphi'}^p]\!]^{\overline{\eta}^p}_{\mathcal{T}}$ for all interpretations η. As a first step, note that if $[\![\varphi']\!]^{\eta[X \mapsto g]}_{\mathcal{T}} \sqsubseteq_\tau g$, then $\overline{g}^s \sqsubseteq_\tau [\![\overline{\varphi'}^p]\!]^{\overline{\eta}^p[\widetilde{X} \mapsto \overline{g}^s]}_{\mathcal{T}}$ since pseudo-complementation inverts order. Hence, f' is in the set

  $$L = \{g \in [\![\tau]\!]_{\mathcal{T}} \mid [\![\varphi']\!]^{\eta[X \mapsto g]}_{\mathcal{T}} \sqsubseteq_\tau g\}$$

  if and only if $\overline{f'}^s$ is in the set

  $$G = \{g \in [\![\tau]\!]_{\mathcal{T}} \mid g \sqsubseteq_\tau [\![\overline{\varphi'}^p]\!]^{\overline{\eta}^p[\widetilde{X} \mapsto g]}_{\mathcal{T}}\}.$$

  Moreover, again by order inversion, if f is the least element of L, then $\overline{f}^s$ is the greatest element of G. The case for greatest fixpoints is proved similarly.

$\square$

3.1.2 Negation Normal Form

HFL admits negation normal form (NNF), i.e., for every formula there is an equivalent formula of the same order in which negation appears only in front of propositions. In particular, negation does not appear in front of lambda variables. This might be expected since $\mathcal{L}_\mu$, which forms the base of HFL, also admits negation normal form. However, the effects of the lambda-calculus part of HFL make it harder to obtain a procedure to convert a formula into negation normal form. For example, consider the formula

$$(\lambda(f : \bullet \to \bullet).\, f\, \langle a \rangle P)\, \lambda(x : \bullet).\, \neg x$$

where the semantics of the formula are not important. Clearly, it is not in NNF since a negation occurs in front of a lambda variable. In this case it is not hard to see that β-reduction yields a formula, namely $\neg\langle a \rangle P$ that is easy to convert to NNF. However, once negation is entangled with fixpoint definitions, it becomes much harder to derive a principle to convert any HFL formula into an equivalent formula in NNF. Let $\tau = \bullet \to \bullet$ and consider, for example, the formula

$$\Big(\mu(X : \tau \to \bullet).\, \lambda(f : \tau).\, (f\, P) \vee \big((X\, (\lambda(x : \bullet).\, f\, (f\, x))\big)\Big)\, (\lambda(y : \bullet).\, \neg\langle a \rangle y).$$

It unfolds to

$$(\neg\langle a \rangle P) \vee (\neg\langle a \rangle \neg\langle a \rangle P) \vee (\neg\langle a \rangle \neg\langle a \rangle \neg\langle a \rangle \neg\langle a \rangle P) \vee \cdots$$

and while it is still possible to find an equivalent formula in NNF, it is possible to come up with even more contrived examples, especially using functions of order higher than 1. However, these two examples already show that it is not quite straightforward to expect HFL to admit negation normal form. However, Lozes [67] was able to show that it is indeed possible. The driving idea behind this is to change the formula such that every function definition, be it via lambda abstraction or via fixpoint definition, expects each argument twice now, once in a positive form (to be made precise), and once in a negative form. For example, the formula $\lambda(x^+ : \bullet).\, \varphi$ would be changed to $\lambda(x^+ : \bullet).\, \lambda(\widetilde{x}^+ : \bullet).\, \varphi'$ The idea then is that x holds the argument, and $\widetilde{x}$ holds its negation, while φ' is the continuation of this principle. At the ground type level, this means that negation boils down to inverting the arguments, respectively choosing the correct one. For example, the function $\lambda(x^- : \bullet).\, \neg x$ changes to

$$\lambda(x^+ : \bullet).\, \lambda(\widetilde{x}^+ : \bullet).\, \widetilde{x}.$$

On higher-order functions this generalizes accordingly.

Formally, negation normal form (NNF) is defined as follows:

Definition 3.1.5. An HFL-formula φ is in *negation normal form* if φ can be derived from the following grammar:

$$\begin{aligned}\varphi ::= P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid x \mid \varphi\, \varphi \mid \lambda(x^+ : \tau).\, \varphi \\ \mid X \mid \mu(X : \tau).\, \varphi \mid \nu(X : \tau).\, \varphi\end{aligned}$$

where $P \in \mathcal{P}$, $a \in A$, $X \in \mathcal{X}$, $x \in \mathcal{F}$ and $\tau \in \mathsf{Types}$.

Note that negation occurs only in front of propositions, but not in front of lambda or fixpoint variables. Moreover, the only permitted variance is $+$. Hence, when dealing with formulas in NNF, we omit variance symbols altogether, since they are always clear from context.

We now present a version of the construction, following the idea of [67]. As a first step, associate to each type τ another type $\mathsf{n}(\tau)$ which uses only the variance $+$, defined via

$$\begin{aligned}\mathsf{n}(\bullet) &= \bullet\\ \mathsf{n}(\tau_1^v \to \tau_2) &= \mathsf{n}(\tau_1)^+ \to \mathsf{n}(\tau_1)^+ \to \mathsf{n}(\tau_2)\end{aligned}$$

Example 3.1.6. If $\tau = (\bullet^- \to \bullet)^+ \to \bullet$, then

$$\mathsf{n}(\tau) = (\bullet^+ \to \bullet^+ \to \bullet)^+ \to (\bullet^+ \to \bullet^+ \to \bullet)^+ \to \bullet.$$

Given an HFL formula φ with fixpoint variables in $\mathcal{X}$, let $\mathcal{X}' = \{X_{\mathsf{nv}} \mid X \in \mathcal{X}\}$ be another set of fixpoint variables. The only purpose of this shift in the variable set is to distinguish the variables used in the negation normal form of a formula from those used in the formula itself. This is useful for the correctness proof. For practical purposes, one can use $X_{\mathsf{nv}} = X$.

Definition 3.1.7. Let φ be a well-typed HFL formula. Define $\mathsf{nnf}(\varphi)$ inductively via

$$\begin{aligned}\mathsf{nnf}(P) &= P\\ \mathsf{nnf}(\neg P) &= \neg P\\ \mathsf{nnf}(\varphi_1 \vee \varphi_2) &= \mathsf{nnf}(\varphi_1) \vee \mathsf{nnf}(\varphi_2)\\ \mathsf{nnf}(\varphi_1 \wedge \varphi_2) &= \mathsf{nnf}(\varphi_1) \wedge \mathsf{nnf}(\varphi_2)\\ \mathsf{nnf}(\langle a\rangle\varphi) &= \langle a\rangle\mathsf{nnf}(\varphi)\\ \mathsf{nnf}([a]\varphi) &= [a]\mathsf{nnf}(\varphi)\\ \mathsf{nnf}(\neg\varphi) &= \mathsf{nnf}(\overline{\varphi}^p) \text{ if } \varphi \neq P\\ \mathsf{nnf}(\varphi_1\,\varphi_2) &= (\mathsf{nnf}(\varphi_1)\,\mathsf{nnf}(\varphi_2))\,\mathsf{nnf}(\overline{\varphi_2}^p)\\ \mathsf{nnf}(\lambda(x^v\colon \tau).\,\varphi) &= \lambda(x^+\colon \mathsf{n}(\tau)).\,\lambda((\widetilde{x})^+\colon \mathsf{n}(\tau)).\,\mathsf{nnf}(\varphi)\\ \mathsf{nnf}(x) &= = x\\ \mathsf{nnf}(\widetilde{x}) &= \widetilde{x}\\ \mathsf{nnf}(\sigma(X\colon \tau).\,\varphi) &= \sigma(X_{\mathsf{nv}}\colon \mathsf{n}(\tau)).\,\mathsf{nnf}(\varphi)\\ \mathsf{nnf}(X) = \mathsf{nnf}(\widetilde{X}) &= X_{\mathsf{nv}}\end{aligned}$$

Note that $\overline{\varphi}^p$ denotes the pseudo-complement of φ introduced in Definition 3.1.1.

Remark 3.1.8. It might appear that $\mathsf{nnf}(\varphi)$ is not well-defined since fixpoint variables are not duplicated, even though they can appear freely in both of the operand sides of an application. For example, one might believe that the formula $\mu(X\colon \bullet).\,\neg\big((\lambda(y^-\colon \bullet).\,\neg y)\,X\big)$ has negation normal form

$$\mu(X_{\mathsf{nv}}\colon \bullet).\,(\lambda(\widetilde{y}^+\colon \bullet).\,\lambda(y^+\colon \bullet).\,y)\,\widetilde{X_{\mathsf{nv}}}\,X_{\mathsf{nv}}$$

in which $\widetilde{X_{\mathsf{nv}}}$ appears freely. However, note that $\widetilde{X_{\mathsf{nv}}}$ is not actually defined since fixpoint variables always appear as their positive variant. This is justified since the HFL typing system ensures that fixpoint variables are never used in negative variance, whence the second operand of the application is irrelevant to the semantics of the application since the variable $\widetilde{y}$ never appears in the operator. The negation normal form of the above formula is actually

$$\mu(X_{\mathsf{nv}} : \bullet).\, (\lambda(\widetilde{y}^{+} : \bullet).\, \lambda(y^{+} : \bullet).\, y)\, X_{\mathsf{nv}}\, X_{\mathsf{nv}}$$

which is well defined.

Lemma 3.1.9. *Let φ be a well-typed* HFL *formula. Then $\mathsf{nnf}(\varphi)$ is also well-typed, of the same order as φ and in NNF. Moreover, for each subformula ψ of φ, if the free lambda variables of ψ are $x_1, \dots, x_n$, then the free lambda variables of $\mathsf{nnf}(\psi)$ are among $x_1, \widetilde{x_1}, \dots, x_n, \widetilde{x_n}$. If the free fixpoint variables of ψ are $X_1, \dots, X_m$, then the free fixpoint variables of $\mathsf{nnf}(\psi)$ are*[1] *among $(X_1)_{\mathsf{nv}}, \dots, (X_m)_{\mathsf{nv}}$.*

Proof. A proof that $\mathsf{nnf}(\varphi)$ is well-typed can be derived from the proof that φ is well-typed by replacing each occurrence of a type τ in the original proof by $\mathsf{n}(\tau)$. The only notable adaption occurs at formula application, where the rule for positive variance has to be used in all cases, and has to be repeated twice, once for each copy of an argument. A minor adaption occurs at negations, where a proof for well-typedness of the pseudo-complemented subformula in question has to be used.

The statements on the free variables are by a simple induction over the syntax tree of φ, and the statement on type order is immediate. Clearly, $\mathsf{nnf}(\varphi)$ is in NNF, since on the right side of the translation, negations occur only in front of propositions, and only the positive variance is used. $\square$

It remains to prove that a closed formula of ground type is actually equivalent to its NNF. Before we do that, we introduce semantic equivalence classes in both type lattices as well as for interpretations. This is due to the fact that, during the inductive correctness proof for NNF, we work with under-defined functions. Passing to equivalence classes solves this problem.

Definition 3.1.10. Let $\mathcal{T}$ be an LTS. For each type τ and each $f \in [\![\tau]\!]_{\mathcal{T}}$, define a subset of $[\![\mathsf{n}(\tau)]\!]_{\mathcal{T}}$ denoted by $[f]$, respectively $[x]$, inductively as

$$\begin{aligned} [f : \bullet] &= \{f\} \\ [f : \tau_1 \to \tau_2] &= \{f' \in [\![\mathsf{n}(\tau_1) \to \mathsf{n}(\tau_1) \to \mathsf{n}(\tau_2)]\!]_{\mathcal{T}} \\ &\qquad \text{s.t. f.a. } g \in [\![\tau_1]\!]_{\mathcal{T}} \text{ if } g_1 \in [g], g_2 \in [\overline{g}^s] \text{ then } \big((f'\, g_1)\, g_2\big) \in [f\, g]\}. \end{aligned}$$

Moreover, given an interpretation η, define a set of interpretations $[\eta]$ via $\eta' \in [\eta]$ if, for each lambda variable x, we have that

$$\begin{aligned} &\eta'(x) \in [\eta(x)] \text{ and} \\ &\eta'(\widetilde{x}) \in [\overline{\eta(x)}^s], \end{aligned}$$

and similarly for fixpoint variables.

[1]The parentheses serve no purpose besides separating the double subscript.

The intuition here is that a function is in $[f\colon \tau_1 \to \tau_2]$ if it behaves suitably like f on argument pairs that are pseudo-complements of each other, but can behave arbitrarily on arguments that do not have this property. Note that, for function types, $f \notin [f\colon \tau_1 \to \tau_2]$ since f is of type $\tau_1 \to \tau_2$, but $[f\colon \tau_1 \to \tau_2] \subseteq [\![\mathsf{n}(\tau_1 \to \tau_2)]\!]_{\mathcal{T}}$. Interpretations in $[\eta]$ are likewise required to map pairs of variables of the form $x, \widetilde{x}$ to pseudo-complementary objects.

Observation 3.1.11. The classes defined in Definition 3.1.10 are each nonempty and compatible with suprema and infima in the sense that if $f'_\beta\, g \in [f_\beta\, g]$ for all $\beta < \alpha$, then $f'_\alpha\, g \in [f_\alpha\ g]$ if $f'_\alpha\, g$ is defined as $\bigsqcap_{\beta<\alpha}(f'_\beta\, g)$ and $f_\alpha\, g$ is defined as $\bigsqcap_{\beta<\alpha}(f_\beta\, g)$, and similarly for $\bigsqcup$.

Lemma 3.1.12. *Let φ be an* HFL *formula of ground type. Then* $\mathsf{nnf}(\varphi) \equiv \varphi$.

Proof. Let $\mathcal{T}$ and η be arbitrary. We now show the following by induction over the syntax tree of φ: For all $\eta' \in [\eta]$, we have that if $[\![\varphi]\!]^{\eta}_{\mathcal{T}} = f$, then $[\![\mathsf{nnf}(\varphi)]\!]^{\eta'}_{\mathcal{T}} \in [f]$ and $[\![\mathsf{nnf}(\overline{\varphi}^p)]\!]^{\eta'}_{\mathcal{T}} \in [\overline{f}^s]$. The claim of the lemma then follows.

- If φ is P or $\neg P$ or a lambda or fixpoint variable, then the claim is immediate, if necessary by invocation of the definition of $[\eta]$. For the cases of boolean and modal operators, the claim follows by a simple application of the induction hypothesis.
- If φ is $\varphi_1\, \varphi_2$, then, by the induction hypothesis, we have that $[\![\mathsf{nnf}(\varphi_i)]\!]^{\eta'}_{\mathcal{T}} \in [[\![\mathsf{nnf}(\varphi_i)]\!]^{\eta}_{\mathcal{T}}]$ for $i \in \{1, 2\}$. Hence, by the definition of $[\cdot]$, we have that $[\![\mathsf{nnf}(\varphi_1)\, \mathsf{nnf}(\varphi_2)]\!]^{\eta'}_{\mathcal{T}} \in [[\![\varphi_1\, \varphi_2]\!]^{\eta}_{\mathcal{T}}]$.
- If φ is $\lambda(x\colon \tau_1).\, \varphi'$, let τ_2 be the type of φ'. By the induction hypothesis, we have that $[\![\mathsf{nnf}(\varphi')]\!]^{\eta'}_{\mathcal{T}} \in [[\![\varphi']\!]^{\eta}_{\mathcal{T}}]$. In particular, for $f \in [\![\tau_1]\!]_{\mathcal{T}}$ such that $f_1 \in [f]$ and $f_2 \in [\overline{f}^s]$, we have that $\eta'[x \mapsto f_1, \widetilde{x} \mapsto f_2] \in [\eta]$, and hence that $[\![\mathsf{nnf}(\varphi')]\!]^{\eta'[x\mapsto f_1, \widetilde{x}\mapsto f_2]}_{\mathcal{T}} \in [[\![\varphi']\!]^{\eta[x\mapsto f]}_{\mathcal{T}}]$. Hence, since $\mathsf{nnf}(\varphi)$ is equal to $\lambda(x\colon \mathsf{n}(\tau_1)).\, \lambda(\widetilde{x}\colon \mathsf{n}(\tau_1)).\, \mathsf{nnf}(\varphi')$, we have that $[\![\mathsf{nnf}(\varphi)]\!]^{\eta'}_{\mathcal{T}} \in [[\![\varphi]\!]^{\eta}_{\mathcal{T}}]$.
- If φ is $\mu(X\colon \tau_1 \to \tau_2).\, \varphi'$, then $\mathsf{nnf}(\varphi) = \mu(X_{\mathsf{nv}}\colon \mathsf{n}(\tau_1 \to \tau_2)).\, \mathsf{nnf}(\varphi')$. By the Kleene Fixpoint Theorem (Theorem 2.1.7), there are ordinals α and α' such that $[\![\varphi]\!]^{\eta}_{\mathcal{T}} = [\![X^\alpha]\!]^{\eta}_{\mathcal{T}}$ and $[\![\mathsf{nnf}(\varphi)]\!]^{\eta'}_{\mathcal{T}} = [\![X^{\alpha'}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}}$. Let $\alpha'' = \max\{\alpha, \alpha''\}$. We now show by induction over the ordinals up to α'' that $[\![X^{\alpha''}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} \in [[\![X^{\alpha''}]\!]^{\eta}_{\mathcal{T}}]$.
  - Note that $[\![X^0]\!]^{\eta}_{\mathcal{T}} = \bot_{\tau_1\to\tau_2}$, while $[\![X^0_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} = \bot_{\mathsf{n}(\tau_1)\to\mathsf{n}(\tau_1)\to\mathsf{n}(\tau_2)}$ and, hence
  $$[\![X^0_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} = \bot_{\mathsf{n}(\tau_1)\to\mathsf{n}(\tau_1)\to\mathsf{n}(\tau_2)} \in [\bot_{\tau_1\to\tau_2}] = [[\![X^0]\!]^{\eta}_{\mathcal{T}}].$$
  - Let β be an ordinal. Now assume that we have shown that, if $[\![X^\beta]\!]^{\eta}_{\mathcal{T}} = f$, then
  $$f' = [\![X^\beta_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} \in [[\![X^\beta]\!]^{\eta}_{\mathcal{T}}] = [f].$$
  Note that $[\![X^{\beta+1}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} = [\![\mathsf{nnf}(\varphi')]\!]^{\eta'[X_{\mathsf{nv}}\mapsto f']}_{\mathcal{T}}$. Since $\widetilde{X_{\mathsf{nv}}}$ does not occur in $\mathsf{nnf}(\varphi')$ due to typing reasons, we can extend $\eta'[X_{\mathsf{nv}} \mapsto f]$ to $\eta'[X_{\mathsf{nv}} \mapsto f, \widetilde{X_{\mathsf{nv}}} \mapsto \overline{f}]$, and we still have
  $$g' = [\![X^{\beta+1}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} = [\![\mathsf{nnf}(\varphi')]\!]^{\eta'[X_{\mathsf{nv}}\mapsto f]}_{\mathcal{T}} = [\![\mathsf{nnf}(\varphi')]\!]^{\eta'[X_{\mathsf{nv}}\mapsto f, \widetilde{X_{\mathsf{nv}}}\mapsto \overline{f}]}_{\mathcal{T}}$$

On the other hand, let g be defined as $[\![X^{\beta+1}]\!]^{\eta}_{\mathcal{T}} = [\![\varphi']\!]^{\eta[X\mapsto f]}_{\mathcal{T}}$. By the induction hypothesis, $g' \in [g]$, which is the claim for the successor ordinal $\beta + 1$.

– Let β be a limit ordinal and suppose that we have shown the claim for all ordinals smaller than β, i.e., for all $\beta' < \beta$, we have that

$$f'_{\beta'} = [\![X^{\beta'}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} \in [[\![X^{\beta'}]\!]^{\eta}_{\mathcal{T}}] = [f_{\beta'}].$$

Let $g \in [\![\tau_1]\!]_{\mathcal{T}}$ be arbitrary. We have that if $g_1 \in [g]$ and $g_2 \in [\overline{g}^s]$, then $f'_{\beta'}\, g'_1\, g_2 \in [f_{\beta'}\, g]$. Now $[\![X^{\beta}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}}\, g_1\, g_2 = \bigsqcup_{\beta'<\beta}(f'_{\beta'}\, g_1\, g_2)$, and $[\![X^{\beta}]\!]^{\eta}_{\mathcal{T}}\, x = \bigsqcup_{\beta'<\beta}(f_{\beta'}\, g)$. By the compatibility of limits with $[\cdot]$ seen in Observation 3.1.11, we obtain that $[\![X^{\beta}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}}\, g_1\, g_2 \in [[\![X^{\beta}]\!]^{\eta}_{\mathcal{T}}\, x]$, and, since g was arbitrary, also that $[\![X^{\beta}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} \in [[\![X^{\beta}]\!]^{\eta}_{\mathcal{T}}]$.

The argument for greatest fixpoints is completely analogous. It follows that $[\![X^{\alpha''}_{\mathsf{nv}}]\!]^{\eta'}_{\mathcal{T}} \in [[\![X^{\alpha''}]\!]^{\eta}_{\mathcal{T}}]$ and, hence that $[\![\mathsf{nnf}(\varphi)]\!]^{\eta'}_{\mathcal{T}} \in [[\![\varphi]\!]^{\eta}_{\mathcal{T}}]$.

□

Theorem 3.1.13. *Each* HFL *formula φ is equivalent to a formula* $\mathsf{nnf}(\varphi)$ *of the same type order and in NNF. The size of* $\mathsf{nnf}(\varphi)$ *is at most exponential in the size of φ.*

The statement on the at most exponential blowup can be easily seen from the definition of the translation. The only place where blowup appears is at function application, and the size at most doubles there. Lozes [67] states his translation to be at most quadratic, but there is a problem in the translation given that argument duplication appears only at operators of variance $\pm$, which can easily be seen as inadequate by introducing dummy negations. However, this does not mean that a correct translation necessarily has exponential blowup.

Conjecture 3.1.14. *Each* HFL *formula φ is equivalent to one in negation normal form of the same type order and of size polynomial in the size of φ.*

3.1.3 Automaton Normal Form

We now introduce a normal form for HFL formulas that is both closer to the syntax of APKA, the automaton model studied in Chapter 4, and eliminates certain features from HFL formulas that are permitted by the HFL syntax, but are hard to manage in some situations. A central feature of *automaton normal form* is that it couples lambda abstraction and fixpoint binders in the sense that

- lambda abstraction can only occur directly after a fixpoint binder, or after another lambda abstraction,
- fixpoint definitions do not have free lambda variables, i.e., formulas of the form $\lambda x.\, \mu X.\, \varphi$ such that x occurs in φ are not permitted,
- the defining formula of a fixpoint, after excluding the fixpoint quantifier itself and the initial string of lambda abstractions, is of ground type.[2]

[2]Note that this is not quite the same as being in η normal form [45]. However, η-expansion will be used to bring formulas into a form adhering to this criterion.

The effect of this is that each fixpoint definition can mostly be considered as a closed unit where dependencies to other fixpoint definitions are reduced to a minimum, exercised through appearance of a fixpoint variable of a fixpoint definition in the defining formula of another fixpoint. In this case, all parameters of the former fixpoint are passed on through application in the defining formula of the latter fixpoint.

Definition 3.1.15. An HFL formula φ of ground type is in *automaton normal form* (ANF) if it satisfies the following conditions:

1. φ is in negation normal form and well-named,
2. φ is of the form $\sigma_{X_I}.(X_I : \bullet).\, \psi_{X_I}$ where $\sigma \in \{\mu, \nu\}$,
3. The defining formula φ_X of each fixpoint variable X of type $\tau_X = \tau_1^X \to \cdots \to \tau_{k_X}^X \to \bullet$ is of the form

   $$\varphi_X = \sigma_X(X : \tau_X).\, \lambda(x_1^X : \tau_1^X). \cdots \lambda(x_{k_X}^X : \tau_{k_X}^X).\, \psi_X$$

   where $\sigma_X \in \{\mu, \nu\}$ and ψ_X is derived from the grammar

   $$\psi ::= P \mid \overline{P} \mid \psi \vee \psi \mid \psi \wedge \psi \mid \langle a \rangle \psi \mid [a]\psi \mid \psi\, \psi \mid x \mid Y \mid \varphi_Y$$

   where $x \in \{x_1^X, \ldots, f_{k_X}^X\}$ and $Y \in \mathcal{X}$.

Note that the symbol φ_Y in the grammar of the last point refers to the defining formula of the fixpoint variable Y. Also note that Y in Item 3 of Definition 3.1.15 is necessarily from the set $\{Y \mid Y \succ X\} \cup \{X\} \cup \{Y \mid X = \mathit{upVar}(Y)\}$.

Example 3.1.16. Consider the formula φ from Example 2.4.6 defined as

$$\Big(\mu(F : \tau \to \bullet).\, \lambda(g : \tau).\, g\, P \vee \big(F\, \lambda(y : \bullet).\, g\, (g\, y)\big)\Big)(\lambda(z : \bullet).\, \langle a \rangle z)$$

where $\tau = \bullet \to \bullet$. It is not in ANF, since lambda abstraction appears freely, and the top operator is not a fixpoint. However, it can be converted into ANF by replacing $\lambda(y : \bullet).\, g\, (g\, y)$ by

$$\big(\mu(G : \tau \to \bullet \to \bullet).\, \lambda(g' : \tau).\, \lambda(y : \bullet).\, g'\, (g'\, y)\big)\, g$$

and by replacing $\lambda(z : \bullet).\, \langle a \rangle z$ by $\mu(S : \bullet \to \bullet).\, \lambda(z : \bullet).\, \langle a \rangle z$ and, finally, by adding a dummy outermost fixpoint of ground type. Note that, in the first replacement formula, the variable g has to be requantified as g' in order to not violate the requirement that fixpoint definitions have no free lambda variables.

We now show that ANF is a proper normal form, i.e., that for each HFL formula, there is an equivalent formula in ANF.

Let φ be an HFL formula of ground type, without loss of generality in negation normal form and well-named. Recall that we can assume that all subformulas of φ are annotated by their respective type. Consider the following function fpmask which masks lambda abstractions not directly below other lambda abstractions or a fixpoint definition with a dummy fixpoint. fpmask takes three arguments: a subformula of φ, $\sigma \in \{\mu, \nu\}$, which signals the polarity of the next fixpoint above in the syntax tree of the formula that fpmask returns, and a bit in $\{0,1\}$ which stores whether the subformula immediately above is either a fixpoint quantifier or a lambda abstraction. If we are not interested in the value of said bit, we just write b in order to avoid needless case duplication.

$$\begin{aligned}
\mathsf{fpmask}(P,\sigma,b) &= P\\
\mathsf{fpmask}(\neg P,\sigma,b) &= \neg P\\
\mathsf{fpmask}(\psi_1 \vee \psi_2,\sigma,b) &= \mathsf{fpmask}(\psi_1,\sigma,0) \vee \mathsf{fpmask}(\psi_2,\sigma,0)\\
\mathsf{fpmask}(\psi_1 \wedge \psi_2,\sigma,b) &= \mathsf{fpmask}(\psi_1,\sigma,0) \wedge \mathsf{fpmask}(\psi_2,\sigma,0)\\
\mathsf{fpmask}(\langle a\rangle\psi,\sigma,b) &= \langle a\rangle\mathsf{fpmask}(\psi,\sigma,0)\\
\mathsf{fpmask}([a]\psi,\sigma,b) &= [a]\mathsf{fpmask}(\psi,\sigma,0)\\
\mathsf{fpmask}(\psi_1\,\psi_2,\sigma,b) &= \mathsf{fpmask}(\psi_1,\sigma,0)\,\mathsf{fpmask}(\psi_2,\sigma,0)\\
\mathsf{fpmask}(((\lambda(x\colon\tau_1).(\psi\colon\tau_2))\colon\tau_1\to\tau_2),\sigma,1) &= \lambda(x\colon\tau_1).\,\mathsf{fpmask}(\psi,\sigma,1)\\
\mathsf{fpmask}(((\lambda(x\colon\tau_1).(\psi\colon\tau_2))\colon\tau_1\to\tau_2),\sigma,0) &= \sigma(X\colon\tau_1\to\tau_2).\,\lambda(x\colon\tau_1).\\
&\qquad\qquad \mathsf{fpmask}(\psi,\sigma,1),\ \text{ where } X \text{ is new}\\
\mathsf{fpmask}(\sigma' X.\,\psi,\sigma,b) &= \sigma' X.\,\mathsf{fpmask}(\psi,\sigma',1)
\end{aligned}$$

The behavior of fpmask is illustrated in Example 3.1.16 where two unmasked lambda abstractions are masked by dummy fixpoint variables.

Lemma 3.1.17. *Let φ be a well-named* HFL *formula of ground type in negation normal form. Let $\varphi' = \mathsf{fpmask}(\varphi,\mu,0)$ or $\varphi' = \mathsf{fpmask}(\varphi,\nu,0)$. Then $\varphi' \equiv \varphi$ and φ' is still well-named, in negation normal form and of the same order as φ.*

Proof. The proof is by induction over the syntax tree of φ. We omit the details since fpmask commutes with all operators with one exception: If a subformula ψ of φ is of the form $\lambda(x\colon\tau_1).\,(\psi'\colon\tau_2)$ such that the operator above ψ is neither a fixpoint binder nor another lambda abstraction, then ψ is replaced by $\sigma(X\colon\tau_1\to\tau_2).\,\lambda(x\colon\tau_1).\,\psi'$ with σ depending on the first fixpoint binder above ψ. Since X is new and, hence, does not occur in ψ, the semantics of HFL yield that $\sigma(X\colon\tau_1\to\tau_2).\,\psi\equiv\psi$. The claim on the order is by straightforward verification. ☐

Note that the second parameter of the transformation, i.e., the polarity of the next fixpoint operator above the subformula in question, is not necessary in order to mask all lambda abstractions by fixpoint operators. However, by remembering said polarity, the translation is conservative with respect to the nesting of least- and greatest-fixpoint operators (cf. Chapter 6) in the sense that it does not add additional nesting, even if only in a vacuous manner.

The next step in transforming an HFL formula into ANF is to ensure that the defining formula of each fixpoint definition is of ground type, i.e., that all parameters of a fixpoint definition are explicitly abstracted away. For example, consider the (non-ground type) formula

$$\mu(X\colon\bullet\to\bullet\to\bullet).\,\nu(Y\colon\bullet\to\bullet\to\bullet).\,\lambda(x\colon\bullet).\,X\,x,$$

where both X and Y have implicit arguments. However, we can make them explicit by first changing the definition of Y such that the formula is

$$\mu(X\colon\bullet\to\bullet\to\bullet).\,\nu(Y\colon\bullet\to\bullet\to\bullet).\,\lambda(x\colon\bullet).\,\lambda(y\colon\bullet).\,(X\,x)\,y,$$

and then updating the definition of X in order to get the formula

$$\mu(X\colon\tau).\,\lambda(x_1\colon\bullet).\,\lambda(x_2\colon\bullet).\,\Big(\big(\nu(Y\colon\tau).\,\lambda(x\colon\bullet).\,\lambda(y\colon\bullet).\,(X\,x)\,y\big)\,x_1\Big)x_2,$$

where $\tau=\bullet\to\bullet\to\bullet$.

This principle is know as η-expansion and is used to bring lambda-calculus terms into so-called η normal form [45]. It can be made formal as follows: Let φ be a well-named HFL formula of ground type in negation normal form, let $\mathcal{X}$ be the set of fixpoint variables of φ and for $X\in\mathcal{X}$, let $\sigma_X X.\,\varphi_X$ be the defining formula of the fixpoint variable X.

Given $X\in\mathcal{X}$ of type $\tau_1\to\cdots\to\tau_n\to\bullet$ and $\varphi_X=\lambda(x_1\colon\tau_1).\cdots\lambda(x_i\colon\tau_i).\,\psi_X$ with $i\le n$ and such that the top operator of ψ_X is not a lambda abstraction, let $\mathcal{Y}=\{Y\mid Y\prec X$ and there is not Z with $Y\prec Z\prec X\}$. Then $\mathsf{makeGT}(\varphi_X)$ is defined as

$$\mathsf{makeGT}(\varphi_X)=\lambda(x_1\colon\tau_1).\dots\lambda(x_i\colon\tau_i).\lambda(x_{i+1}\colon\tau_{i+1}).\dots\lambda(x_{k_X}\colon\tau_{k_X}).\,\psi'_X$$

where

$$\psi'_X=(\cdots(\psi_X[\mathsf{makeGT}(\sigma_Y Y.\,\varphi_Y)/\sigma_Y Y.\,\varphi_Y\mid Y\in\mathcal{Y}]\,x_{k_X})\cdots)\,x_{i+1}.$$

Example 3.1.18. Consider the formula

$$\mu(X\colon\tau\to\bullet\to(\bullet\to\bullet)).\,\lambda(f\colon\tau).\,\lambda(x\colon\bullet).\,f\,x$$

where $\tau=\bullet\to\bullet\to\bullet$. The inner part of the defining formula, namely $f\,x$ of X, is not of ground type. By replacing it with $\lambda(\bullet\colon y).\,(f\,x)\,y$, the new formula $(f\,x)\,y$ is of ground type.

Lemma 3.1.19. *Let φ be a well-named* HFL *formula of ground type in negation normal form, let X be a fixpoint variable in φ such that the defining formula for X is $\sigma_X X.\,\varphi_X$. Then $\sigma_X X.\,\mathsf{makeGT}(\varphi_X)\equiv\sigma_X X.\,\varphi_X$ and both formulas are of the same order.*

Proof. The proof is by induction over $\succ$, i.e., over the fixpoint variables. Assume that the result has been proved for all Y such that $Y\succ X$. Hence, it remains to show that

$$\psi_X\equiv\lambda(x_{i+1}\colon\tau_{i+1}).\dots\lambda(x_{k_X}\colon\tau_{k_X}).\,(\cdots(\psi_X\,x_{k_X})\cdots)\,x_{i+1},$$

which is immediate by invariance of HFL semantics under β-reduction. The claim on the order is by a straightforward verification. ☐

The final step in the process of converting an HFL formula into ANF is to eliminate free lambda variables in fixpoint definitions. For example, consider the formula $\psi=\lambda(x\colon\tau_1).\,\mu(X\colon\tau_2).\,\varphi$ such that x occurs in φ. We can eliminate the free occurrence of x by passing the parameter explicitly to the defining formula of X. The formula

$$\lambda(x\colon\tau_1).\,\big(\mu(X'\colon\tau_1\to\tau_2).\,\lambda(x'\colon\tau_1).\,\varphi[x'/x,(X'\,x')/X]\big)\,x$$

can be shown to be equivalent to ψ. We renamed the fixpoint variable X to X' to avoid confusion since the two variables do not have the same type. The formula above emulates the free occurrence of x in φ by adding another parameter of the type of x to the definition of X', passing x as the actual value of this parameter and then replacing all references of x in φ by x'. This makes x' have the value of x even though formally, it is a different variable. Note that also all occurrences of X need to be amended to $X'\,x'$ in order to pass on the value of x, although it is never changed. Note that this construction can change the order of the fixpoint variable X' if $\mathit{ord}(x) \geq \mathit{ord}(X)$. However, this construction will not change the order of the full formula, since if x is abstracted, then the whole formula will be of order at least $\mathit{ord}(x) + 1$.

We formalize this principle as follows: Let φ be a well-named HFL formula in negation normal form that is of the form $\sigma_{X_I} X_I.\, \varphi_{X_I}$. Let $X_1, \dots, X_n$ be an enumeration of the fixpoint variables in φ compatible with $\succ$ in the sense that $X_{i+j} \not\succ X_i$ for all $1 \leq i \leq n$ and $1 \leq j \leq n-i$. Note that, in particular, $X_1 = X_I$. We define a sequence of formulas $\varphi = \varphi_1, \dots, \varphi_n$ as follows: Let $\{(x_1^i \colon \tau_1^i), \dots, (x_{m_i}^i \colon \tau_{m_i}^i)\}$ be the set of free lambda variables of the subformula $\sigma_{X_i}.(X_i \colon \tau_i^X).\, \psi_i$ in φ_i. Define ψ'_{i+1} as

$$\lambda(y_1^i \colon \tau_1^i).\, \cdots \lambda(y_{m_i}^i \colon \tau_{m_i}^i).\, \psi_i[(\cdots(X'_i\, y_{m_i}^i)\cdots)y_1^i/X_i, y_1^i/x_1^i, \dots, y_{m_i}^i/x_{m_i}^i]$$

and φ_{i+1} as

$$\varphi_i\Big[\Big((\cdots((\sigma_{X_i}(X'_i \colon \tau_1^i \to \cdots \to \tau_{m_i}^i \to \tau_i^X).\, \psi'_{i+1})x_{m_1}^i)\cdots x_1^i)\Big)/\sigma_{X_i}X_i.\, \psi_i\Big],$$

and define $\mathsf{makexExpl}(\varphi)$ as $\mathsf{makexExpl}(\varphi) = \varphi_n$.

We have seen an implicit use of $\mathsf{makexExpl}$ in Example 3.1.16 where the variable g had to be rebound into g'.

Lemma 3.1.20. *Let φ be a well-named* HFL *formula in negation normal form that is of the form $\sigma_{X_I} X_I.\, \varphi_{X_I}$. Then $\varphi \equiv \mathsf{makexExpl}(\varphi)$, and the two formulas are of the same type-theoretic order.*

Proof. The proof is by induction over the ordering of the fixpoint variables defined above. Assuming the result has been proved for all $i' < i$, we have to show that

$$\sigma_{X_i} X_i.\, \psi_i \equiv (\cdots((\sigma_{X_i}(X_i \colon \tau_1 \to \cdots \to \tau_{m_i} \to \tau_i^X).\, \psi'_{i+1})x_1)\cdots x_1).$$

We show the result for $\sigma_{X_i} = \mu$ and $m_i = 1$, in order to improve readability. The proof for $\sigma_{X_i} = \nu$ is completely symmetric, and the proof for $m_i > 1$ proceeds analogously. We also do not display the index i, since it is fixed for the rest of the proof. With these assumptions, we have to show that

$$\mu(X \colon \tau).\, \psi \equiv \big(\mu(Y \colon \tau' \to \tau).\, \lambda(y \colon \tau').\, \psi[Y\, y/X, y/x]\big)\, x.$$

Fix an arbitrary LTS $\mathcal{T}$ and some interpretation η that interprets x. Let

$$g = [\![\mu(X \colon \tau).\, \psi]\!]_{\mathcal{T}}^{\eta} = \bigsqcap\{d \in [\![\tau]\!]_{\mathcal{T}} \mid [\![\psi]\!]_{\mathcal{T}}^{\eta[X \mapsto d]} \sqsubseteq_{\tau} d\}$$

and let

$$\begin{aligned} g' &= [\![\mu(X' \colon \tau' \to \tau).\, \lambda(y \colon \tau').\, \psi[Y\, x'/X, y/x]]\!]_{\mathcal{T}}^{\eta} \\ &= \bigsqcap\{d' \in [\![\tau' \to \tau]\!]_{\mathcal{T}} \mid [\![\lambda(y \colon \tau').\, \psi[Y\, x'/X, y/x]]\!]_{\mathcal{T}}^{\eta[X' \mapsto d']} \sqsubseteq_{\tau' \to \tau} d'\} \end{aligned}$$

By the semantics of HFL, it remains to show that $g'\,\eta(x) = g$, which we will do by showing that $g'\,\eta(x) \sqsubseteq_\tau g$ and $g \sqsubseteq_\tau g'\,\eta(x)$. The latter is obtained by showing that $g'\,\eta(x)$ is actually a fixpoint of the monotone operator $d \mapsto [\![\psi]\!]_{\mathcal{T}}^{\eta[X\mapsto d]}$. For this, consider $g'\,\eta(x)$, for which the equation

$$g'\,\eta(x) = [\![\lambda(y\colon \tau').\, \psi[X'\,y/X, y/x]]\!]_{\mathcal{T}}^{\eta[X'\mapsto g']}\,\eta(x)$$

holds because g' is a fixpoint. By invariance of HFL semantics under β-reduction, this reduces to

$$[\![\psi[Y\,x'/X, y/x]]\!]_{\mathcal{T}}^{\eta[X'\mapsto g', y\mapsto\eta(x)]}.$$

Since y only appears as substitution instance of x, by invariance of HFL semantics under renaming, this further simplifies to

$$[\![\psi[X'\,x/X]]\!]_{\mathcal{T}}^{\eta[X'\mapsto g']} = [\![\psi]\!]_{\mathcal{T}}^{\eta[X\mapsto g'\,\eta(x)]}$$

which yields that $g'\,\eta(x)$ is a fixpoint of the operator $d \mapsto [\![\psi]\!]_{\mathcal{T}}^{\eta[X\mapsto d]}$. Since g is the least such fixpoint, we have that $g \sqsubseteq_\tau g'\,\eta(x)$.

Conversely, consider the formula $\lambda x.\, \mu X.\, \psi$, where $x\colon \tau'$ holds. It is a fixpoint of the operator

$$d' \mapsto [\![\lambda y.\, \psi[X'\,y/X, y/x,]]\!]_{\mathcal{T}}^{\eta[X'\mapsto d']},$$

which we show as follows. Let $g'' = [\![\lambda x.\, \mu X.\, \psi]\!]_{\mathcal{T}}^{\eta}$. Then

$$[\![\lambda y.\, \psi[X'\,y/X, y/x]]\!]_{\mathcal{T}}^{\eta[X'\mapsto g'']}$$

equals

$$[\![\lambda y.\, \psi[y/x, \big((\lambda x.\, \mu X.\, \psi)\,y\big)/X]]\!]_{\mathcal{T}}^{\eta}$$

if we replace g'' by its definition. Using β-reduction, this equals

$$[\![\lambda y.\, \psi[y/x, \big(\mu X.\, \psi[y/x]\big)/X]]\!]_{\mathcal{T}}^{\eta}$$

which, by invariance of HFL under renaming, is the same as

$$[\![\lambda x.\, \psi[\big(\mu X.\, \psi\big)/X]]\!]_{\mathcal{T}}^{\eta}.$$

By the fixpoint unfolding principle, i.e., the fact that $[\![\mu X.\, \psi]\!]_{\mathcal{T}}^{\eta} = [\![\psi[\mu X.\, \psi/X]]\!]_{\mathcal{T}}^{\eta}$, this is the same as g''. But since $[\![\lambda x.\, \mu X.\, \psi]\!]_{\mathcal{T}}^{\eta}$ is a fixpoint of the operator

$$d' \mapsto [\![\lambda y.\, \psi[Y\,x'/X, y/x]]\!]_{\mathcal{T}}^{\eta[X'\mapsto d']}$$

of which g' is the least fixpoint, we have that $g'\,f \sqsubseteq_\tau g''\,f$ for all $f \in [\![\tau']\!]_{\mathcal{T}}$. This holds in particular for $\eta(x)$, whence $g'\,\eta(x) \sqsubseteq_\tau [\![(\lambda x.\, \mu X.\, \psi)\,x]\!]_{\mathcal{T}}^{\eta}\,\eta(x) = [\![\mu X.\, \psi]\!]_{\mathcal{T}}^{\eta[x\mapsto\eta(x)]} = [\![\mu X.\, \psi]\!]_{\mathcal{T}}^{\eta} = g$.

Regarding the order, note that each step does not change the order of the substituted formula. It is possible that the order of individual fixpoint variables increases, but the additional arguments to these fixpoint variables are stand-ins for the lambda variables that occurred freely in their defining formula before the substitution. Hence, the order of the complete formula stays unchanged. $\square$

Note that this procedure can increase the arity of the types of the fixpoint variables involved considerably, in particular for the lowest fixpoints with respect to the fixpoint variable order $\prec$.

We are now ready to prove that ANF is actually a proper normal form.

Lemma 3.1.21. *For every* HFL *formula φ of ground type and in NNF, there is an equivalent formula φ' of the same order that is in ANF and of size at most polynomial in the size of φ.*

Proof. Without loss of generality, φ is well-named. If the top operator of φ is not a fixpoint binder, replace φ by $\sigma(X_I : \bullet).\, \varphi$. Clearly, this retains semantics. In the following, we will assume that φ already has this format.

Let $\varphi' = \mathsf{fpmask}(\varphi)$. By the last clause of the definition of fpmask we have that φ' is of the form $\sigma_{X_I} X_I.\, \varphi'_{X_I}$. Let $\varphi'' = \sigma_{X_I} X_I.\, \mathsf{makeGT}(\varphi'_{X_I})$ and let $\varphi''' = \mathsf{makexExpl}(\varphi'')$. We claim that φ''' is in ANF, has the same type-theoretic order as φ and, moreover, is equivalent to φ.

The equivalence claim and the claim on order is by application of Lemmas 3.1.17, 3.1.19 and 3.1.20. In order to show the claim that φ''' is in ANF, first note that φ''' is still in negation normal form since none of the procedures applied introduces any negations. Moreover, note that $\varphi' = \mathsf{fpmask}(\varphi)$ is such that lambda abstraction can only appear after another lambda abstraction or after a fixpoint binder. However, the defining formulas of fixpoints, if excluding the initial string of lambda abstractions, are not necessarily of ground type, and the fixpoint formula potentially has free lambda variables. However, neither makeGT nor $\mathsf{makexExpl}$ introduce lambda abstractions except immediately after the sequence of lambda abstractions following a fixpoint binder, or between the binder and the sequence of abstractions. Hence, φ''' has the property that lambda abstractions appear only directly after fixpoint binders or other lambda abstractions.

As a second step, note that the defining formula of a fixpoint after the sequence of lambda abstractions is of ground type after using makeGT, and notice that $\mathsf{makexExpl}$ does not change that. Finally, $\varphi''' = \mathsf{makexExpl}(\varphi'')$ is such that fixpoint formulas do not have free lambda variables by construction. Hence, the defining formula of a fixpoint definition follows the format of Item 3 of Definition 3.1.15. It follows that φ''' is in ANF.

The statement on the size of the blowup is by a straightforward verification that the individual steps do not produce more than a polynomial blowup. □

Observation 3.1.22. If φ is in ANF, then $\overline{\varphi}^p$ also is in ANF.

This follows from the fact that pseudo-complementation just reverses the polarity of all operators.

3.2 The Model-Checking Game

We now introduce the model-checking game mentioned at the beginning of the chapter. Remember that this game is intended to give operational semantics to HFL and needs to work even on infinite structures. Hence, the game itself is infinite and, even over finite structures, almost all of its plays are infinite as well.

For ease of exposition, we fix an HFL formula in ANF for the remainder of the section in order to avoid repeatedly making assumptions on it. Let φ be a well-named HFL formula in ANF and let $\mathcal{T}, v_I$ with $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ be a pointed LTS. Let $\mathcal{X}$ be the set of φ's fixpoint variables, and let X_I be the outermost of them. Let τ_X be the type of X for $X \in \mathcal{X}$. Moreover, let $\varphi_X = \sigma_X X.\, \lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X$ be the defining formula of X for $X \in \mathcal{X}$. Note that the only fixpoint variables that occur freely in φ_X are those from the set $\{Y \mid Y \succ X\}$, and φ_X has no free lambda variables since φ is in ANF. Note that $\varphi = \varphi_{X_I}$ by the definition of ANF.

μ-Signatures

A μ-signature s is a mapping that associates an ordinal to each least-fixpoint variable that occurs in φ. The purpose of a signature is to map each least-fixpoint variable to an approximation in the sense of the Kleene Fixpoint Theorem (Theorem 2.1.7). This allows us to replace the least-fixpoint semantics in HFL formulas, which is non-constructive in the sense of the Knaster-Tarski Theorem (Theorem 2.1.5), by the somewhat more constructive semantics of Kleene. Note that, contrary to standard implementations [84] of the concept of μ-signatures, we do not fix a total order of the fixpoint variables in question. The technique of μ-signatures goes back to Emerson and Street [84].

If X is a fixpoint variable and s is a μ-signature, the update $s[X \mapsto \alpha]$ is defined as

$$s[X \mapsto \alpha](Y) = \begin{cases} \alpha \text{ if } Y = X \\ s(Y) \text{ otherwise.} \end{cases}$$

Definition 3.2.1. Let s and s' be two μ-signatures and let $X \in \mathcal{X}$. We say that s' is *descending* from s with respect to X if

- X is a greatest-fixpoint variable and $s(Y) \geq s'(Y)$ for all $Y \succ X$, or
- X is a least-fixpoint variable, $s(Y) \geq s'(Y)$ for all $Y \succ X$, and $s(X) > s'(X)$.

Lemma 3.2.2. *Let $(s_i)_{i\in\mathbb{N}}$ and $(X_i)_{i\in\mathbb{N}\setminus\{0\}}$ be sequences of μ-signatures and fixpoint variables from $\mathcal{X}$ such that X_i and X_{i+1} are comparable with respect to $\succ$, and such that s_{i+1} is descending from s_i with respect to X_{i+1} for all $i > 0$. Then there is a unique highest fixpoint variable X with respect to $\prec$ that occurs infinitely often in the sequence of fixpoint variables. Moreover, X is a greatest-fixpoint variable.*

Proof. Since $\mathcal{X}$ is finite, at least one fixpoint variable must occur infinitely often in $(X_i)_{i\in\mathbb{N}}$. Since any two subsequent fixpoint variables in $(X_i)_{i\in\mathbb{N}\setminus\{0\}}$ are comparable by $\succ$, between each two incomparable such variables there must be an occurrence of a variable that is comparable to both. This variable must be outermore than both by the definition of $\succ$. Hence, if two incomparable variables occur infinitely often, there is another variable greater than both that also occurs infinitely often. It follows that there is a unique fixpoint variable X that occurs infinitely often in $(X_i)_{i\in\mathbb{N}\setminus\{0\}}$.

For the sake of contradiction, assume that X is a least-fixpoint variable. Since it is the highest fixpoint variable to occur infinitely often in $(X_i)_{i\in\mathbb{N}\setminus\{0\}}$, for some $n > 0$ no fixpoint variable greater than X with respect to $\succ$ occurs. By the definition of being descending with respect to some variable, we notice that $s_{i+1}(X) \leq s_i(X)$

for all $i \geq n$. Moreover, for all $j \geq n+1$ such that $X_j = X$, we have that $s_{j-1}(X) > s_j(X)$. Hence the sequence $(s_j(X))_{X_j = X}$ is an infinitely descending chain of ordinals, which contradicts that the ordinals are well-ordered. Hence, X is a greatest-fixpoint variable. □

For each fixpoint variable X and for each signature s we define semantics of X^s. For greatest-fixpoint variables, this is just the greatest fixpoint itself, while for least-fixpoint variables, this is an approximation of φ_X. In either case, the interpretation for fixpoint variables that are outermore than X is derived from s. This approximation is defined via

$$\begin{aligned}
&[\![X^s]\!]_{\mathcal{T}} = [\![\varphi_X]\!]_{\mathcal{T}}^{\eta^s} && \text{if } \sigma_X = \nu \\
&[\![X^s]\!]_{\mathcal{T}} = [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\,\mathtt{ff}]\!]_{\mathcal{T}} && \text{if } \sigma_X = \mu, s(X) = 0 \\
&[\![X^s]\!]_{\mathcal{T}} = [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\,\varphi'_X]\!]_{\mathcal{T}}^{\eta^{s[X\mapsto\alpha]}} && \text{if } \sigma_X = \mu, s(X) = \alpha+1 \\
&[\![X^s]\!]_{\mathcal{T}} = \bigsqcup_{\alpha < s(X)} [\![X^{s[X\mapsto\alpha]}]\!]_{\mathcal{T}}^{\eta^s} && \text{if } \sigma_X = \mu, s(X) \text{ is a limit ordinal}
\end{aligned}$$

where $\eta^s(Y) = [\![Y^s]\!]_{\mathcal{T}}$.

Lemma 3.2.3. *For each $X \in \mathcal{X}$ and for each μ-signature s, the approximation X^s is well-defined. Moreover, $[\![\varphi_X]\!]_{\mathcal{T}}^{\eta^s} = [\![X^s]\!]_{\mathcal{T}}^{\eta^s}$ if $s(X) = \mathsf{ht}([\![\tau_X]\!]_{\mathcal{T}})$.*

Proof. For the case of a greatest-fixpoint variable, the claim on well-definedness follows since φ_X has as free fixpoint variables only those that are outermore than X. In the second case, well-definedness is immediate. In the third case, well-definedness follows from the fact that $s[X \mapsto \alpha]$ is descending from s with respect to X and the only free fixpoint variables of $\lambda x_1^X.\dots,\lambda x_{k_X}^X.\,\varphi'_X$ are X itself and those that are outermore than X. In the last case, well-definedness follows from well-definedness of the approximations at lower ordinal values for $s(X)$. Note that η^s also defines values for fixpoint variables that are not X nor are outermore than X, but these variables never occur freely in the context of $[\![X^s]\!]_{\mathcal{T}}$.

For greatest-fixpoint variables, the claim on the semantics of X^s is correct by definition. For least-fixpoint variables, note that the definition of $[\![X^s]\!]_{\mathcal{T}}$ mirrors the definition of the approximations in the Kleene Fixpoint Theorem (Theorem 2.1.7), i.e., for each ordinal α we have that $[\![X^s]\!]_{\mathcal{T}}$ is the αth approximation under η^s if $s(X) = \alpha$. Hence, by the Kleene Fixpoint Theorem, the claim for least-fixpoint variables follows. □

The Model-Checking Game

We encode the idea of approximations of the semantics of a fixpoint via a signature syntactically by introducing, for each fixpoint variable in $\mathcal{X}$, and each signature of suitable range, a copy of that variable decorated by the signature. Let VS be a set of fixpoint variables defined via

$$\mathsf{VS} = \{X^s \mid X \in \mathcal{X}, s(Y) \leq \mathsf{ht}([\![\tau_Y]\!]_{\mathcal{T}}) \text{ f.a. } Y \in \mathcal{X}\}$$

where X^s has type τ_X. Since the values of the signature component of the variables are bounded, VS is actually a set. Note that the height of the respective types depends on $\mathcal{T}$, which, however, was fixed at the beginning of the section. Let

$\mathrm{HFL}_{\mathsf{VS}}$ be the set of well-typed[3] HFL formulas in negation normal form with no fixpoint binders, no lambda abstractions, and no free lambda variables, but with free fixpoint variables from VS. In other words, a formula is in $\mathrm{HFL}_{\mathsf{VS}}$ if it can be derived from the grammar

$$\varphi ::= P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \varphi\,\varphi \mid X^s$$

where $X^s \in \mathsf{VS}$.

The intuition for a formula in $\mathrm{HFL}_{\mathsf{VS}}$ is that occurrences of least fixpoints are replaced by a syntactic representation of an approximation to their semantics, indicated by the superscript signature. Greatest fixpoints are similarly decorated by a signature, but this signature only indicates the respective approximations of the least-fixpoint variables contained in the defining formula of the greatest fixpoint. This removes the necessity to have fixpoint binders in the syntax. Since we are starting with a formula in ANF, lambda abstraction occurs only in direct conjunction with a fixpoint binder and, hence, needs no explicit representation in the syntax either.

A position in the game is a triple of the form $(v, \psi, \mathcal{F})$, where

- $v \in S$,
- $\psi \in \mathrm{HFL}_{\mathsf{VS}}$,
- $\mathcal{F}$ is a stack of formulas from $\mathrm{HFL}_{\mathsf{VS}}$ growing from right to left,

such that, if ψ has type $\tau_1 \to \cdots \to \tau_k \to \bullet$, then $\mathcal{F}$ has k elements, it's contents are $\psi_1, \ldots, \psi_k$ from top to bottom and ψ_i is of type τ_i for $1 \leq i \leq k$.

The intuitive meaning of such a triple is that $\mathcal{V}$ tries to prove that $\mathcal{T}, v, \models_{\eta^{\mathsf{VS}}} (\cdots(\psi\,\psi_1)\cdots)\psi_k$, where $\eta^{\mathsf{VS}}(X^s) = [\![X^s]\!]_{\mathcal{T}}$ as defined above for all $X^s \in \mathsf{VS}$. This means that the stack $\mathcal{F}$ contains the arguments at which ψ is to be evaluated, and the fixpoint variables of the form X^s are interpreted as the corresponding approximations as defined above.

The starting position of the game is $(v_I, {X_I}^{s_I}, \varepsilon)$ where s_I is defined via $s_I(Y) = \mathsf{ht}([\![\tau_Y]\!]_{\mathcal{T}})$ for all $Y \in \mathcal{X}$ that are least-fixpoint variables. Note that, in particular, $s_I(X_I) = \mathsf{ht}([\![\tau_{X_I}]\!]_{\mathcal{T}})$ if X_I is a least-fixpoint variable. Clearly, this is a legal position. If a play of the game goes on indefinitely, it is won by $\mathcal{V}$. Note that this makes the model-checking game a *safety game*.

Given a position $(v, \psi, \mathcal{F})$, the game proceeds depending on the form of ψ:

- If ψ is atomic, then $\mathcal{V}$ wins if $\mathcal{T}, v \models \psi$, otherwise, $\mathcal{S}$ wins.
- If ψ is of the form $\psi_1 \vee \psi_2$ then $\mathcal{V}$ picks $i \in \{1, 2\}$ and the game continues from $(v, \psi_i, \mathcal{F})$.
- If ψ is of the form $\psi_1 \wedge \psi_2$ then $\mathcal{S}$ picks $i \in \{1, 2\}$ and the game continues from $(v, \psi_i, \mathcal{F})$.
- If ψ is of the form $\langle a \rangle \psi'$ then $\mathcal{V}$ picks w with $v \xrightarrow{a} w$ and the game continues from $(w, \psi', \mathcal{F})$. If there is no such vertex w, then $\mathcal{V}$ loses.

[3]Here, well-typed means that a typing judgement is derivable if sufficiently many typing hypotheses of the form $X^s \colon \tau_X$ are added. Since a formula can only have finitely many free variables, a suitable collection of such hypotheses forms a valid context.

- If ψ is of the form $[a]\psi'$ then $\mathcal{S}$ picks w with $v \xrightarrow{a} w$ and the game continues from $(w, \psi', \digamma)$. If there is no such vertex w, then $\mathcal{S}$ loses.

- If $\psi = \psi_1\, \psi_2$ then the game continues from $(v, \psi_1, \digamma')$ where $\digamma' = \psi_2 \cdot \digamma$, i.e., $\digamma$ with ψ_2 pushed to the top. This is a legal position since, if ψ has type $\tau_1 \to \cdots \to \tau_k \to \bullet$ and ψ_2 has type τ, then ψ_1 has type $\tau \to \tau_1 \to \cdots \to \tau_k \to \bullet$. Since $\digamma$ contains suitable $\mathrm{HFL}_{\mathsf{VS}}$ formulas of types $\tau_1, \dots, \tau_k$ from top to bottom, we have that $\digamma'$ contains suitable $\mathrm{HFL}_{\mathsf{VS}}$ formulas of types $\tau, \tau_1, \dots, \tau_k$ from top to bottom.

- If $\psi = X^s$, the polarity of X is important. If X is a least-fixpoint variable and $s(X) = 0$, then $\mathcal{V}$ loses the game immediately. Otherwise let α be an ordinal defined as follows: If $s(X)$ is a limit ordinal β then $\mathcal{V}$ picks an ordinal α such that $\alpha < \beta$. If not, then $s(X) = \beta + 1$ for some ordinal β. In this case, set α to be β.

  Let s' be defined via

$$\begin{aligned} s'(Y) &= s(Y) && \text{if } Y \succ X \text{ and } \sigma_Y = \mu \\ s'(Y) &= \alpha && \text{if } Y = X \text{ and } \sigma_X = \mu \\ s'(Y) &= \mathsf{ht}([\![\tau_Y]\!]_{\mathcal{T}}) && \text{if } Y \not\succ X \text{ and } \sigma_Y = \mu \end{aligned}$$

  Let $\psi_1, \dots, \psi_{k_X}$ be the contents of $\digamma$ from top to bottom, let $\mathcal{X}_1 = \{Y \mid Y \succ X\} \cup \{X\}$, and let $\mathcal{X}_2 = \{Y \mid X = \mathit{upVar}(Y)\}$. The game continues with (v, ψ', ε) where

$$\psi' = \varphi'_X[\psi_i/x_i^X \mid 1 \leq i \leq k_X][Y^{s'}/Y \mid Y \in \mathcal{X}_1][Y^{s'}/\varphi''_Y \mid Y \in \mathcal{X}_2]$$

  and[4] $\varphi''_Y = \varphi_Y[Z^{s'}/Z \mid Z \in \mathcal{X}_1]$. Since φ is in ANF, φ'_X is easily seen to be of ground type and to have no free lambda variables. Moreover, since the only free fixpoint variables of φ'_X are those in $\mathcal{X}_1 \cup \mathcal{X}_2$, this formula is in $\mathrm{HFL}_{\mathsf{VS}}$. Hence $(v, \varphi', \varepsilon)$ is a legal position.

Note that, in the fixpoint variable step for some fixpoint variable X, the new signature s' renews the counters for all fixpoint variables that are neither X itself nor above it with respect to $\succ$. In particular, counters for variables that are incomparable with X with respect to $\succ$ are also renewed. It would be equally correct to only renew the counters for variables that are lower than X with respect to $\succ$, or even just for variables in $\mathcal{X}_2$, i.e., those variables Y such that $X = \mathit{upVar}(Y)$ and which, hence, appear in φ'_X. However, the present definition is conceptually the easiest to handle. Note that signatures for fixpoint variables that are substituted into φ'_X from the argument stack are *not* changed. In particular, even counters for variables in $\mathcal{X}_2$ in a signature for a fixpoint variable being substituted from the stack are not renewed. Hence, a fixpoint variable can occur decorated with several distinct signatures in the formula part of a game position.

[4]This definition would not be necessariy if the substitutions for the fixpoint variables in $\mathcal{X}_1$ and $\mathcal{X}_2$ were done in the opposite order. However, doing the substitutions in this order makes the proof of completeness for this game easier.

Example 3.2.4. Let $\mathcal{T} = (\{v\}, \{v \xrightarrow{a} v\}, \emptyset)$ be a one-vertex LTS such that this vertex is reachable from itself via an a-transition. Consider the formula

$$\varphi = \nu(X : \bullet).\, \langle a \rangle \big((\mu(Y : \bullet \to \bullet).\, \lambda(x : \bullet).\, x \vee Y\, x)\, X\big).$$

Note that the height of the lattice $[\![\bullet]\!]_{\mathcal{T}}$ is 2, and the height of the lattice $[\![\bullet \to \bullet]\!]_{\mathcal{T}}$ is 3. Hence, VS can be chosen as $\{Z^s \mid Z \in \{X, Y\}, s \in \{s_0, \dots, s_3\}\}$ where s_i maps Y to i. The starting position of the game is $(v, X^{s_3}, \varepsilon)$. The next position is $(v, \langle a \rangle (Y^{s_3}\, X^{s_3}), \varepsilon)$. Here, $\mathcal{V}$ picks a successor of v for the game to continue in, but since v is the only successor of itself, $\mathcal{V}$ necessarily picks v and the game continues in $(v, Y^{s_3}\, X^{s_3}, \varepsilon)$ and then (v, Y^{s_3}, X^{s_3}) and $(v, X^{s_3} \vee Y^{s_2}\, X^{s_3}, \varepsilon)$. Note that the occurrence of Y is labeled by s_2, i.e., the number of unfoldings allowed for this least-fixpoint variable has decreased. If $\mathcal{V}$ were to pick the left disjunct, the game continues in $(v, X^{s_3}, \varepsilon)$, which is the initial position of the game again. Hence, $\mathcal{V}$ can win this game by always choosing the let disjunct. If, on the other hand, she were to pick the right disjunct, the game would continue in $(v, Y^{s_2} X^{s_3}, \varepsilon)$ and then (v, Y^{s_2}, X^{s_3}) and $(v, X^{s_3} \vee Y^{s_1}\, X^{s_3}, \varepsilon)$. Here, $\mathcal{V}$ can still pick the left disjunct to win the game. If she continues picking the right disjunct, the play continues via the positions $(v, Y^{s_1}\, X^{s_3}, \varepsilon)$, (v, Y^{s_1}, X^{s_3}), $(v, X^{s_3} \vee Y^{s_0}\, X^{s_3}, \varepsilon)$, $(v, Y^{s_0}\, X^{s_3}, \varepsilon)$ and (v, Y^{s_0}, X^{s_3}), at which point $\mathcal{V}$ has lost. However, always picking the left disjunct is, as we have already seen, a winning strategy.

It is important to note that the winning condition of this HFL model-checking game is asymmetric, because the way fixpoint variables are handled differs depending on their polarity. If a play of the game reaches a position of the form $(\_, X^s, \_)$ and X is a least-fixpoint variable, $\mathcal{V}$ loses the game if $s(X) = 0$. Even if this is not the case, the next position will be such that the μ-signature value for X is strictly smaller. Hence, unless she can regenerate the value for X on this variable-signature pair, $\mathcal{V}$ must eventually avoid to unfold it, for otherwise the counter for X will eventually reach 0. On the other hand, $\mathcal{S}$ never loses the game in this way, since greatest-fixpoint variables are not annotated by signatures.

Of course it would be possible to extend signatures in such a way that they also cover greatest-fixpoint variables, and then to extend the winning condition for $\mathcal{V}$ to positions where a position of the form $(\_, X^s, \_)$ is reached where $s(X) = 0$ and X is a greatest-fixpoint variable. This would actually make every play of the game finite, something we do not desire, since the purpose of the model-checking game is to allow $\mathcal{V}$ to construct a winning strategy in the acceptance game for the yet to be defined APKA (cf. Chapter 4) from a winning strategy in the model-checking game of a suitable HFL formula. Since said acceptance game has no such signature mechanics and the winner of an infinite play in this game is decided after the play has concluded, we need the model-checking game to go on indefinitely, even if $\mathcal{V}$ would win a play in the variant where also greatest-fixpoint variables are annotated. Hence, the winning condition of the game assigns all plays to $\mathcal{V}$ if she can avoid defeat indefinitely.

Completeness of the Game

We now prove completeness of the HFL model-checking game, i.e., that $\mathcal{V}$ wins the game from $(v_I, {X_I}^{s_I}, \varepsilon)$ if $\mathcal{T}, v_I \models \varphi$. Soundness of the game, i.e., the converse of this statement will not be proved directly. This is because a proof for this is not quite straightforward. However, soundness of the game will be proved as Corollary 4.3.13 in Section 4.3.

Lemma 3.2.5. *If $\mathcal{T}, v_I \models \varphi$, then $\mathcal{V}$ wins the game from the initial position $(v_I, {X_I}^{s_I}, \varepsilon)$ where X_I is the unique outermost fixpoint variable of φ.*

Proof. The proof will be an induction over the length of a given play showing that $\mathcal{V}$ can avoid losing indefinitely. In preparation for it, define the interpretation η^{VS} with domain VS as $\eta^{\mathsf{VS}}(X^s) = [\![X^s]\!]_{\mathcal{T}}$.

The induction invariant of the proof is that $\mathcal{V}$ can always maintain a position $(v, \psi, \digamma)$ such that $v \in (\cdots([\![\psi]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}\, [\![\psi_1]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}) \cdots [\![\psi_k]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}})$ if the contents of $\digamma$ are $\psi_1, \dots, \psi_k$ from top to bottom.

By Lemma 3.2.3, we have that $[\![{X_I}^{s_I}]\!]_{\mathcal{T}} = [\![\varphi_{X_I}]\!]_{\mathcal{T}} = [\![\varphi]\!]_{\mathcal{T}}$ both in the case that X_I is a least-fixpoint variable and the case that it is a greatest-fixpoint variable. Hence, the starting position satisfies the induction invariant since $\mathcal{T}, v_I \models \varphi$.

Now assume that the game has progressed to some position $(v, \psi, \digamma)$ which satisfies the induction invariant. Depending on the form of ψ, we show how $\mathcal{V}$ can use this to avoid losing and to maintain the induction invariant in the next position, if it exists.

- If ψ is a proposition or a negated proposition, it follows from the invariant that $\mathcal{V}$ wins the game.

- If ψ is of the form $\psi_1 \vee \psi_2$, the argument stack is empty, since $\bullet$ is not a function type. Since $v \in [\![\psi_1 \vee \psi_2]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$, by the definition of HFL semantics, there must be $i \in \{1, 2\}$ such that $v \in [\![\psi_i]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$. Then $\mathcal{V}$ picks i and the game continues in (v, ψ_i, ε), which satisfies the induction invariant.

- If ψ is of the form $\psi_1 \wedge \psi_2$, the argument stack is empty. Since $v \in [\![\psi_1 \wedge \psi_2]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$, by the definition of HFL semantics, $v \in [\![\psi_i]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$ for $i \in \{1, 2\}$. Hence, no matter whether $\mathcal{S}$ picks the left or the right conjunct for the game to continue in (v, ψ_i, ε), the next position satisfies the induction invariant.

- If ψ is of the form $\langle a \rangle \psi'$, the argument stack is empty. Since $v \in [\![\langle a \rangle \psi']\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$, by the definition of HFL semantics, $w \in [\![\psi']\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$ for some vertex w with $v \xrightarrow{a} w$. Then $\mathcal{V}$ picks this vertex w and the game continues in (w, ψ', ε), which satisfies the induction invariant.

- If ψ is of the form $[a]\psi'$, the argument stack is empty. Since $v \in [\![[a]\psi']\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$, by the definition of HFL semantics, $w \in [\![\psi']\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}$ for all vertices w with $v \xrightarrow{a} w$. Hence, no matter what vertex w is picked by $\mathcal{S}$ for the game to continue in, the next position (w, ψ', ε) satisfies the induction invariant.

- If ψ is of the form $\psi_1\, \psi_2$, the next position is $(v, \varphi_1, \digamma')$ where $\digamma' = \psi_2 \cdot \digamma$. Let the contents of $\digamma$ be $\psi'_1, \dots, \psi'_k$ from top to bottom. Then the induction invariant for the next position is that

$$v \in (\cdots(([\![\psi_1]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}\, [\![\psi_2]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}})\, [\![\psi'_1]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}) \cdots [\![\psi'_k]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}})$$

which, by the definition of HFL semantics, equals

$$v \in (\cdots([\![\psi]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}\, [\![\psi'_1]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}}) \cdots [\![\psi'_k]\!]^{\eta^{\mathsf{VS}}}_{\mathcal{T}})$$

which, in turn, was the invariant for $(v, \psi, \digamma)$. Hence, the invariant also holds for the next position.

- If ψ is of the form X^s note that $s(X) \neq 0$ if X is a least-fixpoint variable, for otherwise $\mathcal{V}$ loses the game, which contradicts the induction invariant. If $s(X) = \beta + 1$, set α to be β. Otherwise, $s(X)$ is a limit ordinal β. Note that $\eta^{\mathsf{VS}}(X^s) = [\![X^s]\!]_{\mathcal{T}}$. By the induction invariant, we have $v \in (\cdots([\![X^s]\!]_{\mathcal{T}}\,[\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})\cdots[\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})$ and, hence there must be an ordinal $\alpha < \beta$ such that $v \in (\cdots([\![X^{s[X\mapsto\alpha+1]}]\!]_{\mathcal{T}}\,[\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})\cdots[\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})$. Note that still $\alpha+1 < \beta$. Then $\mathcal{V}$ picks α as part of her move.

  Let s' be the μ-signature from the fixpoint variable case of the game, i.e., let s' be defined via

  $$\begin{aligned} s'(Y) &= s'(Y) && \text{if } Y \succ X \text{ and } \sigma_Y = \mu \\ s'(Y) &= \alpha && \text{if } Y = X \text{ and } \sigma_X = \mu \\ s'(Y) &= \mathsf{ht}([\![\tau_Y]\!]_{\mathcal{T}}) && \text{if } Y \not\succ X \text{ and } \sigma_Y = \mu. \end{aligned}$$

  The next configuration of the game is (v, ψ', ε) with

  $$\psi' = \varphi'_X[\psi_i/x_i \mid 1 \leq i \leq k][Y^{s'}/Y \mid Y \in \mathcal{X}_1][Y^{s'}/\varphi''_Y \mid Y \in \mathcal{X}_2]$$

  where $\psi_1, \ldots, \psi_k$ are the contents of $\digamma$ from top to bottom, and where $\mathcal{X}_1 = \{Y \mid Y \succ X\} \cup \{X\}$ and $\mathcal{X}_2 = \{Y \mid X = \mathit{upVar}(Y)\}$ and $\varphi''_Y = \varphi_Y[Z^{s'}/Z \mid Z \in \mathcal{X}_1]$. Hence, we have to show that

  $$v \in [\![\varphi'_X[\psi_i/x_i \mid 1 \leq i \leq k][Y^{s'}/Y \mid Y \in \mathcal{X}_1][Y^{s'}/\varphi''_Y \mid Y \in \mathcal{X}_2]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}$$

  since this is equivalent to the induction invariant, namely that $v \in [\![\psi']\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}$.

  We begin to show the above by first leaving the elements of the argument stack as arguments rather than substitution instances, and by retaining fixpoint variables from $\mathcal{X}_2$ as proper fixpoint definitions as inherited from φ (which is an HFL formula in ANF rather than an HFL$_{VS}$-formula). By the induction invariant for the previous configuration, we have that

  $$v \in (\cdots([\![X^s]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\,[\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})\cdots)[\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.$$

  Let $\mathcal{X}'_1 = \{Y \mid Y \succ X\}$. Note that

  $$\eta^s(Y) = \eta^{s'}(Y) = [\![Y^{s'}]\!]_{\mathcal{T}} = \eta^{\mathsf{VS}}(Y^{s'})$$

  for $Y \in \mathcal{X}'_1$ since s and s' agree on $Y \succ X$.

  We now show that, in fact,

  $$v \in (\cdots([\![\lambda x_1^X.\ldots\lambda x_{k_X}^X.\,\varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}'_1][X^{s'}/X]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\,[\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})\cdots)[\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}},$$

  which is the same as

  $$v \in (\cdots([\![\lambda x_1^X.\ldots\lambda x_{k_X}^X.\,\varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\,[\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})\cdots)[\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}$$

  First, note that $[\![X^s]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}$ equals $[\![X^s]\!]_{\mathcal{T}}$ by the definition of η^{VS}. For the rest of the claim, there are three cases:

– If X is a greatest-fixpoint variable, then $[\![X^s]\!]_{\mathcal{T}} = [\![\varphi_X]\!]_{\mathcal{T}}^{\eta^s}$. By the fixpoint unfolding principle, we have

$$[\![\varphi_X]\!]_{\mathcal{T}}^{\eta^s} = [\![\nu X.\, \lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X]\!]_{\mathcal{T}}^{\eta^s} = [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X[\varphi_X/X]]\!]_{\mathcal{T}}^{\eta^s}.$$

But since the free fixpoint variables of φ_X are only those that are outermore than X, and since s and s' agree on these variables, we have that $[\![\varphi_X]\!]_{\mathcal{T}}^{\eta^s} = [\![X^{s'}]\!]_{\mathcal{T}}$, whence

$$\begin{aligned}&[\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X[\varphi_X/X]]\!]_{\mathcal{T}}^{\eta^s}\\ &= [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}'_1][X^{s'}/X]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\\ &= [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.\end{aligned}$$

It follows that

$$[\![X_s]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}} = [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.$$

The claim then follows.

– If X is a least-fixpoint variable and $s(X)$ is the successor ordinal $\alpha + 1$, we have

$$[\![X^s]\!]_{\mathcal{T}} = [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X]\!]_{\mathcal{T}}^{\eta^{s[X\mapsto\alpha]}}.$$

Again, since $s[X \mapsto \alpha]$ and s' agree on the free fixpoint variables of φ'_X, namely X and all fixpoint variables that are outermore than X, we obtain that

$$\begin{aligned}&[\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X]\!]_{\mathcal{T}}^{\eta^{s[X\mapsto\alpha]}}\\ =&[\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\end{aligned}$$

which is as desired. It follows that

$$[\![X_s]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}} = [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \varphi'_X[Y_{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.$$

The claim then follows.

– Finally, if X is a least-fixpoint variable and $s(X)$ is a limit ordinal, as seen above $\mathcal{V}$ can pick an ordinal α such that

$$v \in (\cdots([\![X^{s[X\mapsto\alpha+1]}]\!]_{\mathcal{T}}\, [\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}) \cdots [\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}).$$

Similar to the previous case, we derive that

$$\begin{aligned}[\![X^{s[X\mapsto\alpha+1]}]\!]_{\mathcal{T}} &= [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X]\!]_{\mathcal{T}}^{\eta^{s[X\mapsto\alpha]}}\\ &= [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.\end{aligned}$$

Hence, also

$$v \in [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'^X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.$$

In either of the three cases, define ψ'' as

$$\psi'' = \lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1].$$

Now let $\mathcal{X}_2 = \{Y \mid X = upVar(X)\}$. Note that, by definition, for each $Y \in \mathcal{X}_2$, the formula φ_Y occurs in φ'_X. Hence, $\varphi''_Y = \varphi_Y[Z^{s'}/Z \mid Z \in \mathcal{X}_1]$ occurs in ψ''. Note that we have that

$$\begin{aligned}&[\![\varphi_Y[Z^{s'}/Z \mid Z \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\\ =&[\![\sigma_Y Y.\, \lambda x_1^Y.\dots.\lambda x_{k_Y}^Y.\, \varphi'_Y[Z^{s'}/Z \mid Z \in \mathcal{X}_1]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.\end{aligned}$$

Since $s'(Y) = \mathsf{ht}([\![\tau_Y]\!]_{\mathcal{T}})$, by Lemma 3.2.3 this equals $[\![Y^{s'}]\!]_{\mathcal{T}}$ and $\eta^{\mathsf{VS}}(Y^{s'})$. It follows that

$$[\![\psi'']\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}} = [\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1][Y^{s'}/Y \mid Y \in \mathcal{X}_2]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}.$$

Hence, we have that

$$\begin{aligned}v \in (\cdots([\![\lambda x_1^X.,\dots,\lambda x_{k_X}^X.\, \varphi'_X[Y^{s'}/Y \mid Y \in \mathcal{X}_1]&\\ [Y^{s'}/Y \mid Y \in \mathcal{X}_2]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}\, [\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}})\cdots)[\![\psi_k]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}&.\end{aligned}$$

By β-reduction, we obtain that

$$v \in [\![\varphi'_X[\psi_i/x_i \mid 1 \le i \le k][Y^{s'}/Y \mid Y \in \mathcal{X}_1][Y^{s'}/Y \mid Y \in \mathcal{X}_2]]\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}$$

which is nothing else than $v \in [\![\psi']\!]_{\mathcal{T}}^{\eta^{\mathsf{VS}}}$. Since the next position of the game is (v, ψ', ε), the induction invariant also holds for the next position.

Since $\mathcal{V}$ can always avoid losing any play that goes on indefinitely, and outright wins any finite game, she wins any play of the game. This finishes the proof. □

3.3 The Complexity of the HFL Model-Checking Game

We briefly analyze the complexity of deciding the winner of the HFL model-checking game. As a first step, note that, even on finite structures, the game graph of the game is not necessarily finite, because already a single greatest-fixpoint formula can generate an infinite game graph. Consider the order-1 formula

$$\varphi = \nu(Y : \bullet).\, \big(\nu(X : \bullet \to \bullet).\, \lambda(x : \bullet).\, x \wedge X\, \langle a \rangle x\big)\, P$$

which, after discarding the dummy variable Y, unfolds to the infinite conjunction

$$\bigwedge_{i \in \mathbb{N}} \langle a \rangle^i P.$$

Since φ is in ANF, we can analyze its model-checking game over a given LTS. Note that, since only greatest-fixpoint variables occur in φ, all μ-signatures in the game are empty. Hence, we do not display them. For the sake of simplicity, consider the one-vertex LTS $\mathcal{T}, v$ with no actions and no propositions. Already over this LTS, the game generates the infinite sequence of positions

$$(v, Y, \varepsilon), (v, (X\,P), \varepsilon), (v, (P \wedge X\,\langle a \rangle P), \varepsilon), \dots, \\ (v, (\langle a \rangle P \wedge X\,\langle a \rangle^2 P), \varepsilon), \dots, (v, (\langle a \rangle^2 P \wedge X\,\langle a \rangle^3 P), \varepsilon), \dots$$

It is not hard to see that $(v, (\langle a \rangle^i P \wedge X\,\langle a \rangle^{i+1} P), \varepsilon)$ is a reachable position in the game graph for all $i \in \mathbb{N}$.

However, for formulas that do not contain greatest-fixpoint variables, over finite structures the game graph is necessarily finite, since all the type lattices involved have finite height. Moreover, it is, of course, possible to replace the behavior of infinite unfolding for greatest-fixpoints by a signature condition mirroring that for least-fixpoint variables, in which case the game graph becomes finite over finite structures. Since the HFL model-checking game solves[5] the HFL model-checking problem, for formulas of order k, the k-EXPTIME lower bound for HFL model-checking gives a k-fold exponential lower bound for the size of the game graph of the HFL model-checking game. The reason for this bound is that the model-checking game, after possibly modifying it for greatest-fixpoint variables, is a simple alternating reachability game, which can be solved in time polynomial in its size [46]. Hence, the size of the game graph must be of size at least k-fold exponential in the size of the LTS. Furthermore, the length of β-reduction chains of order-k simply-typed-lambda-calculus formulas is known to be potentially k-fold exponentially long [10]. It is therefore natural to expect that the HFL model-checking game has equally large, if not larger game graphs due to the combination of the Simply-Typed Lambda Calculus with fixpoint quantifiers. This intuition is correct as, even in the case of formulas containing only least fixpoints, already very simple formulas can produce a rather extreme blowup.

Let $\tau = (\bullet \to \bullet) \to \bullet$ and let φ be defined as

$$\varphi = \Big(\mu(X : \tau).\, \lambda(f : \bullet \to \bullet).\, f\,P \vee X\,\big(\lambda(x : \bullet).\, f\,(f\,x)\big)\Big)\, \lambda(y : \bullet).\, \langle a \rangle y$$

which can be made to be in ANF by adding a dummy outermost fixpoint state and masking the two unmasked lambda abstractions by dummy fixpoints. We choose not to do so in order to improve readability. By fixpoint unfolding, φ can be seen to be equivalent to the infinite disjunction

$$\bigvee_{i \in \mathbb{N}} \langle a \rangle^{2^i} P$$

and, in fact, partial unfolding such as it happens in the HFL model-checking game will approximate this disjunction. Given an LTS of size n, the height of the lattice of $(\bullet \to \bullet) \to \bullet$ is in the order of $2_2^{p(n)}$ where p is some polynomial [6], whence the infinite disjunction is only generated up to

$$\bigvee_{0 \leq i \leq 2_2^{p(n)}} \langle a \rangle^{2^i} P$$

[5]Remember that soundness of the game was not shown yet, but follows from Corollary 4.3.13 in Section 4.3.

since after that, the signature for X reaches 0. However, this still generates a subformula of the form $\langle a\rangle^{2_3^{p(n)}} P$, and unraveling all the diamonds in this formula actually generates threefold exponentially many subformulas. This is, of course, far worse than the 2-EXPTIME complexity one would expect for solving the model-checking problem of this formula. The blowup becomes even more bizarre over an LTS with two vertices which are connected with each other, but do not loop. Over this LTS, any formula of the form $\langle a\rangle^i P$ is equivalent to $\langle a\rangle^j P$ where $j \equiv i \mod 2$.

The reason for these effects is that the model-checking game works on a purely syntactic level, with no regard for the actual structure of the LTS in question. While, at least for least fixpoints, the restriction on unfolding of fixpoints introduced by the μ-signatures prohibits unnecessary fixpoint unfolding beyond the height of the lattice in question, the formulas generated during this process are far from syntactically minimal representations of the lattice elements they define. This becomes clear in the above example of a two-vertex LTS, where any sequence of diamonds is equivalent to one of size one or two, but where the model-checking game generates a triply exponential chain of diamonds. This problem is in line with observations made for HFL model-checking that it is often beneficial to handle at least some objects during model-checking on a semantic level, i.e., to compute the semantics of a formula in question. This then makes it trivial to compare equivalence of the objects represented by formulas, and, at least in some cases, can lead to complexity theoretic ([63, 21, 20], see Chapter 5 for details) and practical [79] improvements.

We close this discussion by showing that the blowup introduced by the HFL model-checking game can easily be made even more extreme. Recall that $\tau = (\bullet \to \bullet) \to \bullet$. Reconsider the formula

$$\varphi = \Big(\mu(X : \tau).\, \lambda(f : \bullet \to \bullet).\, f\, P \vee X\, \big(\lambda(x : \bullet).\, f\, (f\, x)\big)\Big)\, \lambda(y : \bullet).\, \langle a\rangle y$$

from above. The behavior of this formula can be chained, multiplying the blowup produced. Consider ψ defined as

$$\Big(\mu X.\, \lambda f.\, f\, P \vee X\, \big(\lambda x.\, \big(\mu Y.\, \lambda g.\, \lambda z.\, g\, z \vee Y\big(\lambda z'.\, g\, (g\, z')\big)\big)\, x\big)\Big)\, \lambda y.\, \langle a\rangle y,$$

where X and Y are of type $(\bullet \to \bullet) \to \bullet$ and f and g are of type $\bullet \to \bullet$ and the remaining variables are of ground type. Now, instead of doubling the number of diamonds in its argument f with each recursive call, said argument is run through the copy Y of X before a recursive call. Hence, after the first unfolding of X in the model-checking game and the subsequent unfolding of Y, the next call of X is called with an argument equivalent to

$$\lambda x.\, \bigvee_{0 \le i \le 2_2^{p(n)}} \langle a\rangle^{2^i} x$$

where n is the size of the LTS. For the third unfolding of X, this function is now run through Y again, which increases the length of the longest chain of diamonds encoded in the argument by another three exponentials. Since X can be unfolded two-fold exponentially often itself, this produces a sequence of diamonds whose length is a tower of height twofold exponential in the size of the LTS in question, far beyond any reasonable size of the object represented by it. And, of course, the pattern can be repeated with three or more fixpoint variables, producing an even larger blowup.

Observation 3.3.1. Even for formulas where the game graph of the HFL model-checking problem is finite over finite structures, its size can still be nonelementary in both the size of the structure and the size of the formula.

This observation very clearly signals that the HFL model-checking game is not a good tool to solve the HFL model-checking problem. Rather, it should be understood as a semantic tool to give operational semantics with certain properties to HFL formulas. We use it for exactly this purpose in Chapter 4.

Chapter 4

Alternating Parity Krivine Automata

This chapter is dedicated to the presentation of an automaton model that captures the semantics of HFL. The model, called *Alternating Parity Krivine Automata* (APKA) consists of an extension of ordinary PA (cf. Section 2.2.4) by Krivine's Abstract Machine (cf. Section 2.3.2). Like Krivine's Machine, APKA compute a head normal form of the formula they represent. However, they inherit support for boolean and modal operators from PA, as well as semantics for fixpoints via unfolding to the defining formula. Hence, the computation of an APKA never gets stuck, except at a proposition, in which case the APKA either accepts or rejects.

Infinite computations are resolved by an extended parity condition. As we already outlined in Section 2.4.5, a simple parity condition is not sufficient to capture the semantics of HFL. We borrow the notion of an *unfolding tree* from [62] in order to isolate the part of a computation that represents true infinite recursion from parts that do not. However, contrary to the situation for FLC, it is less obvious that an unfolding tree of an infinite computation actually possesses an infinite path, and that this path is unique. Hence, a long stretch of this chapter is dedicated to establishing this result, which can be thought of as a normalization proof relative to the infinite recursion induced by the fixpoint semantics of APKA. It should be noted that the necessary prioritization of an operand part of an application discussed in Section 2.4.5 can be found in the fact that Krivine's Machine computes head normal forms, which naturally prioritizes the operator over the operand.

The definition of APKA can be found in Section 4.1, while acceptance is discussed in Section 4.2. The chapter closes with translations from HFL to APKA (Section 4.3) and from APKA to HFL (Section 4.4). Both translations follow the general pattern exhibited in the setting of $\mathcal{L}_\mu$ and PA (cf. Section 2.2.5); in particular the direction from an automaton to a formula requires an unraveling argument. Given that the machinery around HFL can be rather unwieldy at times, compared to that surrounding $\mathcal{L}_\mu$, we employ the HFL model-checking game defined in Section 3.2 as an intermediate step between APKA and HFL.

4.1 Syntax

Definition 4.1.1. An *Alternating Parity Krivine Automaton* (APKA) of order k and index m is a five-tuple $(\mathcal{Q}, \Delta, Q_I, \delta, (\tau_Q)_{Q \in \mathcal{Q}})$ where

- $\mathcal{Q}$ is a finite set of *fixpoint states*,
- $\mathcal{F} = \bigcup_{Q \in \mathcal{Q}} \{f_1^Q, \dots, f_{k_Q}^Q\}$ is a finite sets of lambda variables, partitioned into subsets corresponding to a state each,
- $Q_I \in \mathcal{Q}$ is the *initial state*,
- $(\tau_Q)_{Q \in \mathcal{Q}}$ are HFL types of order at most k and of the form $\tau_1^Q \to \cdots \to \tau_{k_Q}^Q \to \bullet$ corresponding to each state such that the type for Q_I is $\bullet$,
- $\Delta \colon \mathcal{Q} \to \mathbb{N}$ is a function called the *priority labeling* with range of size m, and
- δ is a *transition relation* that maps each state to an HFL-formula in negation normal form such that the formula φ_Q for state Q is derived from the following grammar

  $$\varphi ::= P \mid \overline{P} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid (\varphi\,\varphi) \mid f \mid \varphi\,\varphi \mid Q'$$

  where $f \in \{f_1^Q, \dots, f_{k_Q}^Q\}$, $Q' \in \mathcal{Q}$ and, moreover, the judgment

  $$f_1^Q \colon \tau_1^Q, \dots, f_{k_Q}^Q \colon \tau_{k_Q}^Q, Q_1 \colon \tau_{Q_1}, \dots, Q_n \colon \tau_{Q_n} \vdash \varphi_Q \colon \tau_Q$$

  is derivable from the HFL typing system displayed in Figure 2.2 on page 33.

Intuitively, an APKA extends the correspondence between $\mathcal{L}_\mu$ and PA to HFL. Starting from an HFL formula in ANF, the fixpoint states in $\mathcal{Q}$ can be thought of as encoding the semantics of the fixpoint variables in that formula, and the transition relation encodes the defining formula of the fixpoints – note the similarity between the requirements on the transition relation here and the defining formulas for a formula in ANF. However, instead of precedence between fixpoint variables being given implicitly via their position in the syntax tree of a formula, in an APKA, precedence between fixpoint states is given explicitly via a priority labeling, and there is no requirement of having the fixpoint states be partially ordered in any way.

We define the *extended state space* $\mathcal{Q}^e_{\mathcal{A}}$ of $\mathcal{A}$ as $Q \cup \bigcup_{Q \in \mathcal{Q}} \mathrm{sub}(\delta(Q))$. Since most members of the extended state space are subformulas in the transition relation, we use χ, ζ as variables when talking about members of the extended state space, which we call *substates*. Similarly to the definition for subformulas of an HFL formula, we define the *type* of a substate as the type used for this substate in the proof that the APKA in question is well-typed. Since the typing hypotheses are fixed for a given APKA, we do not display them when making statements on the type of a given substate. We call the set $\{f_i^Q \mid 1 \leq i \leq k_Q\}$ the set of lambda variables of Q, with the assumption that, even if the variables are not decorated by their associated fixpoint state, for each of them the fixpoint state in whose transition relation a variable appears is unique. We define the size of an APKA to be the size of the combined subformula set of its transition relation formulas. Note that, similarly to the case of PA, the number of states and the number of priorities are also useful size measures.

If we want to present a concrete[1] APKA, we present it as a finite list of definitions of the form

$$Q \colon (f_1^Q \colon \tau_1^Q, \dots, f_{k_f}^Q \colon \tau_{k_Q}^Q) \mapsto_{\Delta(Q)} \delta(Q).$$

[1]As opposed to a generic APKA in e.g., the prerequisites of a theorem.

Most of the components of an APKA can be deduced from a sequence of such definitions. By convention, the set of fixpoint states is the set for which definitions are given, and their variables and types are uniquely determined by the part before $\mapsto_{\Delta(Q)}$. The index in the latter part uniquely determines the priority labeling, and the part after it gives the transition relation. Given such a list of definitions, it only remains to specify the initial state unless it is clear from context, e.g., because it is the only state of ground type.

Example 4.1.2. Consider the APKA $\mathcal{A}$ defined via $\mathcal{A} = (\mathcal{Q}, \Delta, I, \delta, (\tau_Q)_{Q\in\mathcal{Q}})$ with

- $\mathcal{Q} = \{I, H, F, DS\}$,
- $\Delta(I) = \Delta(H) = \Delta(D) = \Delta(S) = 0, \Delta(F) = 1$,
- $\tau_I = \bullet, \tau_H = \tau \to \bullet, \tau_F = \tau_D = (\tau) \to \bullet \to \bullet, \tau_S = \tau$ with $\tau = \bullet \to \bullet$,

and

$$\begin{aligned}
\delta(I) &= H\,S\\
\delta(H) &= (F\,h)\,(P \wedge H\,h)\\
\delta(F) &= (f\,x) \vee (F\,(D\,f))\,x\\
\delta(D) &= g\,(g\,y)\\
\delta(S) &= \langle a\rangle z.
\end{aligned}$$

We can also present it via the following list of fixpoint definitions:

$$\begin{aligned}
I\colon () &\mapsto_0 H\,S\\
H\colon (h\colon \bullet \to \bullet) &\mapsto_0 (F\,h)\,(P \wedge H\,h)\\
F\colon (f\colon \bullet \to \bullet, x\colon \bullet) &\mapsto_1 (f\,x) \vee (F\,(D\,f))\,x\\
D\colon (g\colon \bullet \to \bullet, y\colon \bullet) &\mapsto_0 g\,(g\,y)\\
S\colon (z\colon \bullet) &\mapsto_0 \langle a\rangle z
\end{aligned}$$

It should be noted that the formal definition of $\mathcal{A}$ can be recovered from this list under the assumption that the first state listed is the initial state. This is the case throughout this thesis.

This APKA $\mathcal{A}$ is a more involved variant of the fixpoint formula from Example 2.4.6 which says that there is a number n such that $\bigvee_{n\in\mathbb{N}} \langle a\rangle^{2^i} P$ holds.

4.2 Acceptance

As an automaton, the semantics of an APKA is defined operationally, and in some sense an APKA behaves mostly as the sum of its parts. Acceptance of APKA is defined via a potentially infinite two-player acceptance game played between $\mathcal{V}$ and $\mathcal{S}$. In this game, $\mathcal{V}$ tries to prove that the automaton in question accepts a given LTS, and $\mathcal{S}$ tries to disprove this. The game behaves similarly to a normal model-checking game by keeping, among other things, a pair of a vertex in the LTS and a substate of the APKA in question for each of its positions, which we call *configuration.* $\mathcal{V}$ then tries to prove that the APKA accepts from this position, while $\mathcal{S}$ tries to prove that it does reject. If the game descends through a disjunction, a modal diamond or one of their duals, semantics are resolved through boolean alternation, i.e., one of the players picks a successor configuration. As one would expect in the acceptance game of an automaton for a fixpoint logic, if the game reaches a substate that is actually a fixpoint state, then the state is simply expanded to its transition relation. This is similar to model-checking games for, e.g., $\mathcal{L}_\mu$, where upon reaching a fixpoint variable, it is expanded to its defining formula.

The previous behavior alone, however, is not sufficient to lift the semantics of a PA to the higher-order features of HFL. In order to close this gap, we borrow the behavior of Krivine's Abstract Machine, which computes the semantics of simply-typed-lambda-calculus expressions. Upon reaching an application substate in the acceptance game, an APKA proceeds by moving the operand part of the application onto a stack designed to hold arguments to higher-order substates, and continues with the operator. Moreover, in analogy to ANF, functions are only defined at fixpoint states, which can be thought of as having a tacit string of lambda abstractions in front of their transition relation. Hence, upon reaching a fixpoint, all elements of the argument stack are consumed and interpreted as being the intended semantics of the lambda variables in the transition relation. The Krivine Machine parts of the APKA then execute the computation of head normal form via call-by-name semantics.

However, in contrast to PA, infinite plays of the acceptance game are not resolved by a simple parity condition, but by a parity condition on an auxiliary structure that can be thought of as separating inconsequential parts of the play from relevant parts. In some sense, this resembles the stair parity conditions used in conjunction with visibly pushdown automata [66].

4.2.1 Closures and Environments

Fix an APKA $\mathcal{A} = (\mathcal{Q}, \Delta, Q_I, \delta, (\tau_Q)_{Q\in\mathcal{Q}})$. During a play of the – yet to be defined – APKA acceptance game, the players encounter positions in which formulas with free lambda variables appear. *Environments* and *closures* define the intended semantics of these free variables via call-by-name substitution. Note that we use the same meta variables for environments and closures that we used in the exposition of Krivine's Machine, even though environments are defined slightly differently.

Environments and closures used in conjunction with an APKA APKA are defined via mutual recursion:

$$\begin{aligned} e, &::= e^0 \mid (f_1^Q \mapsto c_1, \dots, f_{k_Q}^Q \mapsto c_{k_Q}, j) \\ c, c_1, \dots &::= (\chi, e) \end{aligned}$$

where $Q \in \mathcal{Q}$, χ is a substate, and $j > 0$.

The special environment e^0 is the *empty environment*, which serves as the anchor of this mutual recursion and does not define semantics of any variables. An environment e of the form $(f_1^Q \mapsto c_1, \dots, f_{k_Q}^Q \mapsto c_{k_Q}, j)$ defines the intended semantics of the variables $f_1^Q, \dots, f_{k_Q}^Q$ as the closures $c_1, \dots, c_{k_Q}$ and, moreover, stores a unique index j which serves to order all environments during a run. If Q is of ground type and, hence, has no arguments, we present any associated environment as $(\emptyset, j)$ to make clear that no variables are defined. Environment indices are used to make statements on the behavior of APKA, but have no semantic meaning. In fact, the behavior of APKA is defined independently of environment indices. The injective function *index* defined via

$$\begin{aligned} \mathit{index}(e^0) &= 0 \\ \mathit{index}((f_1^Q \mapsto c_1, \ldots, f_{k_Q}^Q \mapsto c_{k_Q}, j)) &= j \end{aligned}$$

returns the index of an environment. The environment e^0 has index 0 by convention.

A closure c of the form (χ, e) resolves the variable that points to it to the substate χ. If this substate has free variables of its own, these are to be resolved according to e.

We inductively define the set of *well-defined* environments and closures. Intuitively, well-defined environments and closures resolve free variables to objects of the correct type and do not yield undefined variables, neither directly nor indirectly. In particular, no circular definitions are possible.

Definition 4.2.1. Well defined environments and closures used in conjunction with $\mathcal{A}$ are defined as follows:

- e^0 is well defined.
- $e = (f_1^Q \mapsto c_1, \ldots, f_{k_Q}^Q \mapsto c_{k_Q}, j)$ is well-defined if, for all $1 \leq i \leq k_Q$, $c_i = (\chi_i, e_i)$ is well-defined, f_i^Q and χ_i have the same type, and $\mathit{index}(e_i) < \mathit{index}(e)$.
- A closure (χ, e) is well-defined if e is well-defined and defines all free lambda variables of χ to closures of the correct type.

We write $\mathit{Clos}(\mathcal{A})$ for the set of all well-defined closures for $\mathcal{A}$.

Note that $\mathit{Clos}(\mathcal{A})$ generally contains many closures with a given index. However, as we see later, during a given run of $\mathcal{A}$, for each index there is at most one closure with this index.

We identify the type of a closure with the type of its substate, i.e., we say $c = (\chi, e)$ has type τ if $\chi\colon \tau$ holds. We also say that the closure c is *in environment* e if $c = (\_\,, e)$.

Example 4.2.2. Let $\mathcal{A}$ be the APKA from Example 4.1.2. Then

$$\begin{aligned} e_1 &= (\emptyset, 1) \\ e_2 &= (h \mapsto (S, e_1), 2) \\ e_3 &= (f \mapsto (h, e_2), x \mapsto (P \wedge H\,h, e_2), 3) \end{aligned}$$

are well-defined environments mapping variables to well-defined closures. The index of e_i is i. Note that e_1 does not assign any variables.

Given an environment

$$e = (f_1^Q \mapsto c_1, \ldots, f_{k_Q}^Q \mapsto c_{k_Q}, \_)$$

and a variable f_i^Q, where Q is arbitrary, define variable lookup as

$$\mathsf{lookup}(f_i^Q, e) = c_i.$$

Example 4.2.3. Consider the environments from Example 4.2.2. Then

$$\mathsf{lookup}(f, e_3) = (h, e_2)$$

Note that variable lookup is undefined for variables not defined by a given environment. This means that, if some variable is not defined by a given environment, then variable lookup will not proceed in any sort of parent environment, which, when it will be defined, will have a different purpose (see Chapter 6). Instead, variable lookup is undefined in this case. However, due to invariants based in well-formedness and maintained during a run of an APKA, variable lookup of undefined variables will never occur. This behavior with respect to variable lookup is a notable difference from the original definition of the Krivine Abstract Machine [60], in which an environment binds a single variable, and variable lookup for all other variables will continue in the parent environment recursively.

4.2.2 Configurations

In addition to the APKA $\mathcal{A}$, we also fix an LTS $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$. Similarly to our convention for HFL and $\mathcal{L}_\mu$, if discussing whether a given APKA accepts an LTS or not, we tacitly assume that the sets of actions and propositions involved match.

A configuration of the APKA acceptance game of $\mathcal{A}$ over $\mathcal{T}$ has the form

$$(v, (\chi, e), \Gamma),$$

where v is a vertex in S, (χ, e) is a closure and Γ is a stack of closures. Our stacks are presented from right to left, i.e., the topmost element on the stack is on the left. The intended semantics in the acceptance game is that $\mathcal{V}$ tries to prove that the automaton accepts $\mathcal{T}, v$ from χ, where free lambda variables in χ are to be resolved according to e and Γ contains the arguments to χ. In particular, Γ is empty if χ is of type $\bullet$. The task of $\mathcal{S}$ is to prove that the automaton does not accept. We identify the *type* of a configuration with the type of its closure, i.e., the type of χ. Depending on the *form* of χ, we might also speak of an application configuration, or a disjunction configuration etc.

During the game, only *well-formed* configurations appear.

Definition 4.2.4. A configuration $(v, (\chi, e), \Gamma)$ is well-formed if the following hold:

1. If $\chi\colon \tau_1 \to \cdots \to \tau_k \to \bullet$, then $\Gamma = (\chi_1, e_1) \cdots (\chi_k, e_k)$ such that $(\chi_i : \tau_i)$ for all $1 \leq i \leq n$.
2. The closure (χ, e) is well-defined, and so are the closures $c_n, \dots, c_1$ on the argument stack.

We write $\mathsf{Conf}(\mathcal{A}, \mathcal{T})$ to denote the set of well-formed configurations over $\mathcal{A}$ and $\mathcal{T}$.

Example 4.2.5. Let $\mathcal{A}$ be the APKA from Example 4.1.2 and let $\mathcal{T}, v_I$ be the LTS from Example 2.2.1 defined as

$$\mathcal{T} = (\mathbb{N}, \{(i, a, i+1) \mid i \in \mathbb{N}\}, \mathcal{L})$$

where $\mathcal{L}(i)$ is $\{P\}$ if $i = 2^m - 2$ for some $m \in \mathbb{N}$ and $\emptyset$ otherwise, and where $v_I = 0$. Then

$$(0, (I, e^0), \varepsilon)$$

and

$$(0, ((F\,h), e_2), (P \wedge H\,h, e_2))$$

are well-defined configurations, where the environments are as in Example 4.2.2. Note that the argument stack for the second configuration contains one entry, namely the closure $(P \wedge H\,h, e_2)$.

4.2.3 The Acceptance Game

The acceptance game between $\mathcal{V}$ and $\mathcal{S}$ for an APKA $\mathcal{A}$ over an LTS $\mathcal{T}$ with designated vertex v_0 is played on $\mathsf{Conf}(\mathcal{A}, \mathcal{T})$. It starts in the configuration $(v_0, (Q_I, e^0), \varepsilon)$, which is clearly well-formed.

The next move from a well-formed configuration $C = (v, (\chi, e), \Gamma)$ depends on the form of χ:

- If χ is P, then $\mathcal{V}$ wins if $\mathcal{T}, v \models P$, and $\mathcal{S}$ wins otherwise.
- If χ is $\overline{P}$, then $\mathcal{V}$ wins if $\mathcal{T}, v \not\models P$, and $\mathcal{S}$ wins otherwise.
- If $\chi = \chi_1 \vee \chi_2$, then $\mathcal{V}$ picks one of the χ_i and the game continues in the configuration $(v, (\chi_i, e), \Gamma)$.
- If $\chi = \chi_1 \wedge \chi_2$, then $\mathcal{S}$ picks one of the χ_i and the game continues in the configuration $(v, (\chi_i, e), \Gamma)$.
- If $\chi = \langle a \rangle \chi'$, then $\mathcal{V}$ chooses w with $v \xrightarrow{a} w$ and the game continues from $(w, (\chi', e), \Gamma)$. If there is no such w, then $\mathcal{V}$ loses the game.
- If $\chi = [a]\chi'$, then $\mathcal{S}$ chooses w with $v \xrightarrow{a} w$ and the game continues from $(w, (\chi', e), \Gamma)$. If there is no such w, then $\mathcal{S}$ loses the game.
- If $\chi = \chi'\chi''$, then the game continues from $(v, (\chi', e), \Gamma')$ where Γ' is $(\chi'', e) \cdot \Gamma$, i.e., Γ with added top element (χ'', e).
- If $\chi = f$ then the game continues from $(v, \mathsf{lookup}(f, e), \Gamma)$.
- If $\chi = Q$ then the game continues from $C' = (v, (\delta(Q), e'), \varepsilon)$, where $e' = (f_1^Q \mapsto c_1, \dots, f_{k_Q}^Q \mapsto c_{k_Q}, e, j)$ if $\Gamma = c_1, \dots, c_{k_Q}$ from top to bottom, and j is the smallest number bigger than any used index.[2] We say that C' is the configuration where e' was defined, that C is the fixpoint configuration associated to e', and that e is the environment associated to C.

We now show that the above definition is actually valid, and does not contain transitions to ill-formed configurations.

[2]We use the environment indices to make statements about the behavior of an APKA. The APKA itself is oblivious to the indices of the environments, hence we do not introduce an explicit internal state to generate the next environment index.

Lemma 4.2.6. *All configurations that appear in a play of the* APKA *acceptance game are well-formed.*

Proof. The proof is by induction over the play. As already stated, the starting configuration $(v_0, (Q_I, e^0), \varepsilon)$ is well-formed. Assume that the lemma has been proved for some configuration $C_j = (\_, (\chi, e), \Gamma)$ already. Depending on the form of χ, we will show that the following configuration, if it exists, is also well-formed.

- If χ is atomic, there is no next configuration and there is nothing to prove.
- If χ is a disjunction, conjunction, a box or a diamond formula, then well-formedness of C_{j+1} follows from well-formedness of C_j.
- If $\chi = \chi' \, \chi''$, then well-formedness of the closures (χ, e) and (χ'', e) follows from well-formedness of the closure $((\chi' \, \chi''), e)$. If the type of χ is $\tau = \tau_1 \to \cdots \to \tau_n \to \bullet$ then by well-formedness of C_j, the closures on Γ are all well-formed and of types $\tau_1, \ldots, \tau_n$, read top to bottom. Then the type of χ' is $\tau' \to \tau$ form some τ', and the type of χ'' is τ', since χ is a substate in the APKA in question and, hence, well-typed. Since the closure (ζ'', e) is the topmost element of the new argument stack, the well-formedness condition on the argument stack is satisfied in C_{j+1}.
- If χ is a lambda variable f, by well-formedness of e we know that $\mathsf{lookup}(f, e)$ is also well-formed. Since neither the stack nor the type of the closure change, C_{j+1} is also well-formed.
- If χ is a fixpoint variable Q, then the closure of C_{j+1} is $(\delta(Q), e')$ which is of type $\bullet$. Since the argument stack is empty in C_{j+1}, the well-formedness condition on the stack is satisfied in C_{j+1}. Moreover, the type of Q is $\tau_1 \to \cdots \to \tau_{k_Q} \to \bullet$ with variables $f_1^Q, \ldots, f_{k_Q}^Q$ of types $\tau_1, \ldots, \tau_{k_Q}$. Finally, Γ contains, from top to bottom well-formed closures $c_1, \ldots, c_{k_Q}$ of types $\tau_1, \ldots, \tau_{k_Q}$. By definition of the transition relation, $e' = (f_1^Q \mapsto c_1, \ldots, f_{k_Q}^Q \mapsto c_{k_Q}, e, j)$ such that j is the smallest number bigger than any used index. Since all these closures are themselves well-defined, the types match, and j is bigger than any index used so far, e is well-defined and, hence, so is C_{j+1}.

□

Example 4.2.7. Let $\mathcal{A}$ be the APKA from Example 4.1.2 and let $\mathcal{T}, v_I$ be the LTS from Example 4.2.5. Then a play of the acceptance game can proceed as shown in Figure 4.1, depending on $\mathcal{V}$'s and $\mathcal{S}$'s choices. We produce a sequence of configurations on the left. New environments are presented on the right in the line of the configuration where they are created.

From the definition of the transition relation, one can see that the APKA acceptance game combines both a generalization of a model-checking game for $\mathcal{L}_\mu$ (cf. [83]) and a variant of the Krivine Abstract Machine (cf. [60]). Boolean and modal operators are modeled by alternation in the game, i.e., one of the players picking a successor configuration. The lambda-calculus parts of HFL are modeled by the Krivine Abstract Machine, and fixpoints are modeled by simple unfolding to their defining formula (i.e., by transition to $\delta(Q)$ for a fixpoint state Q) as well as the acceptance condition, which is given in Definition 4.2.18.

Figure 4.1: An initial play of the acceptance game of an APKA.

$$
\begin{aligned}
C_0 &= (0, (I, e^0), \varepsilon) && \\
C_1 &= (0, (H\,S, e_1), \varepsilon) && e_1 = (\emptyset, 1) \\
C_2 &= (0, (H, e_1), (S, e_1)) && \\
C_3 &= (0, \big((F\,h)(P \wedge H\,h), e_2\big), \varepsilon) && e_2 = (h \mapsto (S, e_1), 2) \\
C_4 &= (0, (F\,h, e_2), (P \wedge H\,h, e_2)) && \\
C_5 &= (0, (F, e_2), (h, e_2) \cdot (P \wedge H\,f, e_2)) && \\
C_6 &= (0, \big(\big((f\,x) \vee (F\,(D\,f))\,x\big), e_3\big), \varepsilon) && e_3 = (f \mapsto (h, e_2), x \mapsto (P \wedge H\,h, e_2), 3) \\
C_7 &= (0, \big(\big((F\,(D\,f))\,x\big), e_3\big), \varepsilon) && \\
C_8 &= (0, (F\,(D\,f), e_3), (x, e_3)) && \\
C_9 &= (0, (F, e_3), (D\,f, e_3) \cdot (x, e_3)) && \\
C_{10} &= (0, \big(\big((f\,x) \vee (F\,(D\,f))\,x\big), e_4\big), \varepsilon) && e_4 = (f \mapsto (D\,f, e_3), x \mapsto (x, e_3), 4) \\
C_{11} &= (0, (f\,x, e_4), \varepsilon) && \\
C_{12} &= (0, (f, e_4), (x, e_4)) && \\
C_{13} &= (0, (D\,f, e_3), (x, e_4)) && \\
C_{14} &= (0, (D, e_3), (f, e_3) \cdot (x, e_4)) && \\
C_{15} &= (0, (g\,(g\,y), e_5), \varepsilon) && e_5 = (g \mapsto (f, e_3), y \mapsto (x, e_4), 5) \\
C_{16} &= (0, (g, e_5), (g\,y, e_5)) && \\
C_{17} &= (0, (f, e_3), (g\,y, e_5)) && \\
C_{18} &= (0, (h, e_2), (g\,y, e_5)) && \\
C_{19} &= (0, (S, e_1), (g\,y, e_5)) && \\
C_{20} &= (0, (\langle a \rangle z, e_6), \varepsilon) && e_6 = (z \mapsto (g\,y, e_5), 6) \\
C_{21} &= (1, (z, e_6), \varepsilon) && \\
C_{22} &= (1, (g\,y, e_5), \varepsilon) && \\
C_{23} &= (1, (g, e_5), (y, e_5)) && \\
C_{24} &= (1, (f, e_3), (y, e_5)) && \\
C_{25} &= (1, (h, e_2), (y, e_5)) && \\
C_{26} &= (1, (S, e_1), (y, e_5)) && \\
C_{27} &= (1, (\langle a \rangle z, e_7), \varepsilon) && e_7 = (z \mapsto (y, e_5), 7) \\
C_{28} &= (2, (z, e_7), \varepsilon) && \\
C_{29} &= (2, (y, e_5), \varepsilon) && \\
C_{30} &= (2, (x, e_4), \varepsilon) && \\
C_{31} &= (2, (x, e_3), \varepsilon) && \\
C_{32} &= (2, (P \wedge H\,h, e_2), \varepsilon) && \\
C_{33} &= (2, (H\,h, e_2), \varepsilon) && \\
&\vdots &&
\end{aligned}
$$

A notable feature of APKA as defined here is that the creation of new environments, which corresponds to lambda abstraction in HFL, is synchronized to the unfolding of fixpoints, similar to the behavior of HFL formulas in ANF in the HFL model-checking game. It is not completely necessary to do this in order to capture HFL in an automaton model. An earlier version of APKA [16] does not have this restriction. Hence, arbitrary HFL formulas can be translated into this version of APKA without going through ANF. We chose to define APKA like this here for three reasons: First, some of the proofs on the nature of the acceptance game (cf. Subsection 4.2.6) become notably easier if environment creation is tied to fixpoint unfolding. Second, if not for this normalization, it would be quite difficult to formalize when exactly a tentative APKA definition is actually well-formed in the sense that all the typing invariants hold throughout a run. The version given in [16] circumvents this by only looking at APKA that are generated directly from a given HFL formula, while the version presented here allows us to define an APKA directly. The third reason why we couple environment creation and fixpoint unfolding is that at type order 1, the version presented here can be modified to behave almost like an automaton for $\mathcal{L}_\mu$, which allows us to prove strictness of the fixpoint alternation hierarchy for order-1 APKA (cf. Chapter 6).

An important characteristic of the APKA acceptance game is that ground-type variables behave slightly differently from other variables: If a play reaches a configuration with closure (f, e) such that f is of ground type, then no configuration later in the play has a closure in e. Intuitively, this is because a ground-type configuration means that the stack is empty and, after reading the ground-type variable, the game ends up in a configuration with an environment of lower index, which can neither directly nor indirectly resolve variables to closures in e. Consequently, no configuration involving this environment can appear later. This also means that the fixpoint associated to e is not a candidate for infinite recursion and, hence, not relevant to the acceptance condition. However, this is only a sufficient, not a necessary condition. We will make this precise in Examples 6.2.13 and 6.2.14.

Lemma 4.2.8. *Let $(C_i)_{i \in \mathbb{N}}$ be a play of the* APKA *acceptance game and let $C_i = (\_, (f, e), \_)$ be a variable configuration such that f is of ground type. Then there is no configuration $C_j = (\_, (\_, e), \_)$ with $j > i$.*

Proof. Let k be the index of e. Let $C_{i+1} = (\_, (\chi, e'), \varepsilon)$ be the configuration that follows C_i. Note that $(\chi, e') = \mathsf{lookup}(f, e)$. Let k' be the index of e'. Note that, because e defines f to resolve to a closure in e', by well-formedness, $k' < k$. Let $\mathcal{E}_{k'}$ be the smallest set of environments that contains e' and, if it contains some environment $e'' = (\dots, x'' \mapsto (\_, e'''), \dots, \_)$ then it also contains e'''. Intuitively, this set contains all environments reachable directly or indirectly from e' via variable lookup. Note that, by definition, any environment in $\mathcal{E}_{k'}$ defines variables to resolve to closures in $\mathcal{E}_{k'}$, and the indices of environments in $\mathcal{E}_{k'}$ are all less then k'. In particular, $e \notin \mathcal{E}_{k'}$. Define a second set of environments $\mathcal{E}_i$ as the set of environments created after C_i. Clearly, $e \notin \mathcal{E}_i$. We will now show the following: All configurations after C_i have closure components and stack elements only in $\mathcal{E}_{k'} \cup \mathcal{E}_i$ and all environments in $\mathcal{E}_{k'} \cup \mathcal{E}_i$ resolve variables exclusively to closures in $\mathcal{E}_{k'} \cup \mathcal{E}_i$. The statement of the lemma follows from the first part of this claim.

Consider C_{i+1}. It has a closure in e', which is in $\mathcal{E}_{k'}$ and it has an empty stack. Moreover, the set of environments in $\mathcal{E}_i$ that have been created up to this configuration is empty by definition. Now assume for some configuration $C_{i'} = (\_, (\chi, e^{i'}), \_)$

with $i' > i+1$ it has been shown that its closure component and all stack contents are in $\mathcal{E}_{k'} \cup \mathcal{E}_i$, and that all environments from $\mathcal{E}_i$ created up to this configuration satisfy the statement above. Then the same also holds for $C_{i'+1}$.

- If χ is a boolean or modal formula, this is clearly true.
- If χ is an application $(\chi'\,\chi'')$, then the closure of $C_{i'+1}$ is $(\chi', e^{i'})$ which satisfies the statement, and the stack contents are the contents of $C_{i'}$ plus the new closure $(\chi'', e^{i'})$, which also satisfies it. Since no new environment was created, the statement also holds for $C_{i'+1}$.
- If χ is a lambda variable, then the closure of $C_{i'+1}$ is $\mathsf{lookup}(\chi, e^{i'})$ which is a closure in $\mathcal{E}_{j'} \cup \mathcal{E}_i$ by assumption. Moreover, the stack contents of $C_{i'+1}$ are the same as for $C_{i'}$. Since no new environment was created, the statement holds for $C_{i'+1}$.
- If χ is a fixpoint variable, then the closure of $C_{i'+1}$ is $(\delta(Q), e^{i'+1})$ for some fixpoint state Q, and $e^{i'+1}$ is a new environment. Since the argument stack contents of $C_{i'}$ are all in environments from $\mathcal{E}_{k'} \cup \mathcal{E}_i$, the new environment $e^{i'+1}$ resolves variables only to closures in $\mathcal{E}_{k'} \cup \mathcal{E}_i$. Hence, the closure $(\delta(Q), e^{i'+1})$ also satisfies the conditions of the statement, and since the argument stack is empty for $C_{i'+1}$, the statement holds for it.

Since $e \notin (\mathcal{E}_{k'} \cup \mathcal{E}_i)$, but all configurations after C_i have closures exclusively from that set, no configuration with a closure in e appears after C_i. $\square$

4.2.4 Unfolding Trees

Since HFL is a fixpoint logic, it seems natural to use a parity condition as the acceptance condition of APKA in order to capture the behavior of nested least and greatest fixpoints. The presence of boolean alternation and the Lambda Calculus, however, complicates matters. As we have already seen in Section 2.4.5, the lambda-calculus part of HFL can effectively produce scenarios where a fixpoint is unfolded infinitely often without exhibiting infinite recursion. Hence, it is not sufficient to just observe the sequence of fixpoint configurations occurring during a play of the APKA acceptance game and decide the winner based on the highest priority that occurs infinitely often. We give an example similar to the one from Section 2.4.5, but in terms of APKA:

Example 4.2.9. Consider the APKA presented by

$$\begin{aligned} I\colon () &\mapsto_0 Y \\ Y\colon () &\mapsto_0 X\,Y \\ X\colon (f\colon \bullet) &\mapsto_1 f \end{aligned}$$

and its unique run on a one-vertex LTS, which is given as

$$\begin{aligned} C_0 &= (\_\,, (I, e^0), \varepsilon) && \\ C_1 &= (\_\,, (Y, e_1), \varepsilon) && e_1 = (\emptyset, 1) \\ C_2 &= (\_\,, (X\,Y, e_2), \varepsilon) && e_2 = (\emptyset, 2) \end{aligned}$$

$$
\begin{array}{ll}
C_3 = (\_\,, (X, e_2), (Y, e_2)) & \\
C_3 = (\_\,, (f, e_3), \varepsilon) & e_3 = (f \mapsto (Y, e_2), 3) \\
C_4 = (\_\,, (Y, e_2), \varepsilon) & \\
C_5 = (\_\,, (X\,Y, e_4), \varepsilon) & e_4 = (\emptyset, 4) \\
C_6 = (\_\,, (X, e_2), (Y, e_4)) & \\
C_7 = (\_\,, (f, e_5), \varepsilon) & e_5 = (f \mapsto (Y, e_4), 5) \\
C_8 = (\_\,, (Y, e_4), \varepsilon) & \\
\vdots &
\end{array}
$$

where we omit the vertex component of the configurations since it is not relevant. Since neither $\mathcal{V}$ nor $\mathcal{S}$ can make any choices in this game, it is not hard to see that it continues like this forever, repeating the cycle of configurations between C_4 and C_8 with different environments. Hence, both X and Y appear infinitely often during the acceptance game, and the highest priority between X and Y is that of X, namely 1. It follows that, if this APKA was equipped with a simple parity condition, it would reject the one-state LTS.

However, it is easy to see that X simply formalizes the identity function, and is not even recursive. Informally, this APKA can be thought of as encoding the HFL-formula $\nu I.\, \nu Y.\, (\mu X.\, \lambda x.\, x)\, Y$, which can be seen to have the one-state LTS as a model.

The solution to this problem is the following principle, which was observed first[3] in [62] in the context of FLC, albeit in a simpler form: For each application, the evaluation through the APKA acceptance game exhibits infinite recursion in at most one of operator and operand. Contrary to the situation in FLC, however, the operand can be evaluated multiple times and evaluation of the operand does not signal immediately that the evaluation of the operator is finished or finite. Moreover, the presence of boolean alternation is another source of complexity compared to classical lambda-calculus settings: Multiple instances of an operand term may appear in vastly different manifestations due to the choices of the players in the play, even though the restriction to a single play, as opposed to the whole game graph, eliminates boolean alternation.

In order to use the principle above to separate parts of the play that belong to the operator part of an application from the parts belonging to an evaluation of the operand, and different evaluations of the operand from each other, we rearrange the sequence of configurations in a tree such that, at an application configuration, the leftmost subtree contains all configurations associated with an evaluation of the operator, and there is one further subtree for each evaluation of the operand. For example, such a tree for the run from Example 4.2.9 would look as displayed in Figure 4.2.

As a first step towards a proper definition of unfolding trees, we formalize the relationship between an application, which increases the argument stack, fixpoint states, which bind the content of the argument stack to variables in a new environment, and the occurrences of these variables themselves. For each environment and each variable defined by it, we need to be able to retrieve the configuration where

[3]Unfolding trees should not be confused with Böhm Trees, which occur in the context of Higher-Order Recursion-Schemes.

Figure 4.2: The upper part of an unfolding tree of the run from Example 4.2.9.

$$\begin{tikzcd}
& (\_\,,(I,e^0),\varepsilon) \arrow[d, dash] & & \\
& (\_\,,(Y,e_1),\varepsilon) \arrow[d, dash] & & \\
& (\_\,,(X\,Y,e_2),\varepsilon) \arrow[dl, dash] \arrow[dr, dash] & & \\
(\_\,,(X,e_2),(Y,e_2)) \arrow[d, dash] & & (\_\,,(Y,e_2),\varepsilon) \arrow[d, dash] & \\
(\_\,,(f,e_3),\varepsilon) & & (\_\,,(X\,Y,e_4),\varepsilon) \arrow[d, dash] \arrow[dr, dash] & \\
& & (\_\,,(X,e_2),(Y,e_4)) \arrow[d, dash] & (\_\,,(Y,e_4),\varepsilon) \arrow[d, dash] \\
& & (\_\,,(f,e_5),\varepsilon) & \vdots
\end{tikzcd}$$

the exact copy of the closure that the variable resolves to was put on the argument stack. So consider an application configuration $C = (\_\,,(\chi\,\zeta, e), \Gamma)$. The following configuration will be of the form $C' = (\_\,,(\chi, e),(\zeta, e)\cdot\Gamma)$ and the closure (ζ, e) stays on the argument stack until a fixpoint configuration $C_i = (\_\,,(Q,\_),\Gamma')$ is reached. In C_{i+1}, a new environment $e' = (f_1^Q \mapsto c_1, \dots, f_j^Q \mapsto (\zeta, e), \dots, f_{k_Q}^Q \mapsto c_{k_Q}, \_)$ is created. We denote this relationship between e', f_j^Q and C by writing $\mathsf{bnode}_{e'}(f_j^Q) = C$. We now make this notion formal:

Definition 4.2.10. Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the APKA acceptance game. Let C_i be the configuration where the environment $e = (f_1^Q \mapsto c_1, \dots, f_{k_Q}^Q \mapsto c_{k_Q}, \_)$ was created. Then $\mathsf{bnode}_e(f_j^Q)$ is defined to be the configuration with the highest index smaller than i such that its argument stack has less than $k_Q - j + 1$ entries.

This rather cumbersome definition is owed to the fact that the same closure can both be bound multiple times in the same environment, and be bound by other environments as well. The important intuition is that $\mathsf{bnode}_e(f)$ points to the configuration when the value of f under e was put on the argument stack, and that this distinguishes different copies of the same closure.

Example 4.2.11. Consider the play from Example 4.2.7 shown in Figure 4.1. In the environment e_6, the variable z resolves to the closure $(g\,y, e_5)$, which was put on the argument stack after C_{15}. Hence, $\mathsf{bnode}_{e_6}(z) = C_{15}$.

Lemma 4.2.12. *Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the* APKA *acceptance game and let $e = (f_1^Q \mapsto c_1, \dots, f_{k_Q}^Q \mapsto c_{k_Q}, \_)$ be an environment that appears in the game and is created in C_i. Then $\mathsf{bnode}_e(f_j^Q)$ is well-defined for all $1 \le j \le k_Q$. Moreover, if $\mathsf{bnode}_{e'}(f') = \mathsf{bnode}_e(f_j^Q)$ for any $1 \le j \le k_Q$, then $e = e'$ and $f' = f_j^Q$. Finally, if $c_j = (\chi, e')$ then the closure of $\mathsf{bnode}_e(f_j^Q)$ is $((\chi'\,\chi), e')$ for some χ', and no fixpoint configurations appear between $\mathsf{bnode}_e(f_j^Q)$ and C_{i-1}.*

Proof. If e is e^0 then there is nothing to prove, so assume that $e \neq e^0$. Recall that C_i is the configuration where e was created and let C_{i-1} be the associated fixpoint configuration. Fix some $1 \le j \le k_Q$. By well-formedness, we know that the argument stack of C_{i-1} contains k_Q many closures and by the definition of the transition relation, we know that they are, top to bottom, $c_1, \dots, c_{k_Q}$. Moreover, any preceding fixpoint configuration, in particular the initial configuration, has an empty argument stack. Since that stack grows by at most one entry per configuration, there must be at least one configuration before C_{i-1} such that the argument stack contains $k_Q - j$ many entries and, hence, there must be a maximal configuration of this kind, so $\mathsf{bnode}_e(f_j^Q)$ is well defined. Moreover, since all following configurations until C_i have at least $k_Q - j + 1$ many entries, neither of them can be a fixpoint configuration. Thus, since the $(k_Q - j + 1)$st entry is c_j in C_i, by backwards induction, this must be the case as well in the configuration following $\mathsf{bnode}_e(f_j^Q)$. Since $\mathsf{bnode}_e(f_j^Q)$ has one entry less on the argument stack, it must be an application configuration and, by the definition of the transition relation, must have a closure of the form $((\chi' \, \chi), e')$.

It remains to prove that if $\mathsf{bnode}_{e'}(f') = \mathsf{bnode}_e(f_j^Q)$ for any $1 \leq j \leq k_Q$, then $e = e'$ and $f' = f_j^Q$. Let $C_{i'}$ be the configuration where e' was created, and let $C_{i'-1}$ be the associated fixpoint configuration. Since no fixpoint configurations occur between $\mathsf{bnode}_e(f_j^Q)$ and C_i, respectively $\mathsf{bnode}_{e'}(f)$ and $C_{i'-1}$, but the respective first configurations agree, we have that $i = i'$ and, hence, $e = e'$. It also follows that $f' = f_j^Q$ since they point to the same closure, which is tied to stack height and, hence variable order. $\square$

The above definition allows us to rearrange the sequence of configurations in a tree, a so-called *unfolding tree*. In an unfolding tree, the sequence of configurations generally follows the leftmost path. However, once it reaches a lambda variable configuration with closure (f, e), the sequence jumps and it continues as a non-leftmost path directly below $\mathsf{bnode}_e(f)$, which encodes that this path corresponds to an evaluation of the operand part of the application that $\mathsf{bnode}_e(f)$ necessarily is. We will see in conjunction with the translation of an APKA into HFL (cf. Sections 4.3 and 4.4) that only two occurrences of a fixpoint configuration that are comparable in this tree form a proper recursive unfolding, while incomparable occurrences are not recursive unfoldings of each other. This isolates the fixpoint recursion, which is relevant to the winning condition of the acceptance game, from artifacts introduced by higher-order features. In fact, Theorem 4.2.17 tells us that there is exactly one infinite path in such an unfolding tree and this path contains the infinite recursion. However, proving this is not quite straightforward. This is mostly due to the fact that unfolding trees are not necessarily finitely branching, precluding a straightforward invocation of Kőnig's Lemma. So in order to prove the result on unfolding trees, we loosen their definition to that of a *generalized unfolding tree*, where we allow the sequence of configurations to continue as a path even after seeing a lambda variable configuration. The precise conditions depend on the type order of the variable in question.

Definition 4.2.13. Let $\pi = (C_i)_{i \in \mathbb{N}}$ be a play of the APKA acceptance game and let $\mathbb{T} \subseteq \mathbb{N}^*$ be a tree with labels in $\bigcup_{i \in \mathbb{N}} \{C_i\}$. We call $\mathbb{T}$ a *generalized unfolding tree* for π if it satisfies the following conditions:

1. For each i, C_i labels exactly one node in $\mathbb{T}$,
2. The root of $\mathbb{T}$ is labeled by C_0,
3. For all $i \in \mathbb{N}$, if C_i labels t, then C_{i+1} labels $t0$ or $C_i = (\_, (f, e), \_, \_)$, there is u labeled by $\mathsf{bnode}_e(f)$, and C_{i+1} labels uj' for some $j' > 0$.

4. For each t and each $i \geq 0$, $j > 0$, if ti is labeled by C and $t(i+j)$ is labeled by C', then $\mathit{index}(C) < \mathit{index}(C')$.

If a node in a generalized unfolding tree is labeled by an application configuration, we call its leftmost subtree the *operator subtree* or *operator branch* and the remaining subtrees its *operand subtrees* or *operand branches*. It is clear from the definition that this is not a misnomer, i.e., that an operator subtree of a node labeled by an application configuration has as its root a node labeled by a configuration that contains the operator of said application in its closure, and that an operand subtree of a node labeled by an application configuration has as its root a node labeled by a configuration that contains the operand of that application in its closure. Note that operand branches are naturally ordered from left to right by the index of the configuration that labels their respective roots.

Example 4.2.14. Let $\mathcal{A}$ be the APKA from Example 4.1.2, let $\mathcal{T}, v_I$ be the LTS from Example 4.2.5 and let $(C_i)_{i\in\mathbb{N}}$ be the play of the acceptance game of $\mathcal{A}$ from $\mathcal{T}, v_I$ that was sketched in Example 4.2.7. The tree in Figure 4.3 is the upper part of a generalized unfolding tree of this play. For space constraints, only the number of the configuration in question and the substate of the closure component are shown.

We observe that generalized unfolding trees are not necessarily finitely branching: if the play continues, more and more sons of the node labeled $1 : H\,S$ will be generated, corresponding to unfoldings of S.

A generalized unfolding tree is called an *unfolding tree* if the sequence of configurations jumps after every variable node, i.e., if variable nodes are always leaves. The tree in Example 4.2.14 is an unfolding tree. Since the proof of Theorem 4.2.17 depends heavily on type order of the variables where the sequence does or does not jump, we formalize this by calling a generalized unfolding tree *reduced down to type level k* if the sequence of configurations jumps at a lambda-variable configuration if and only if its type is of order k or less.

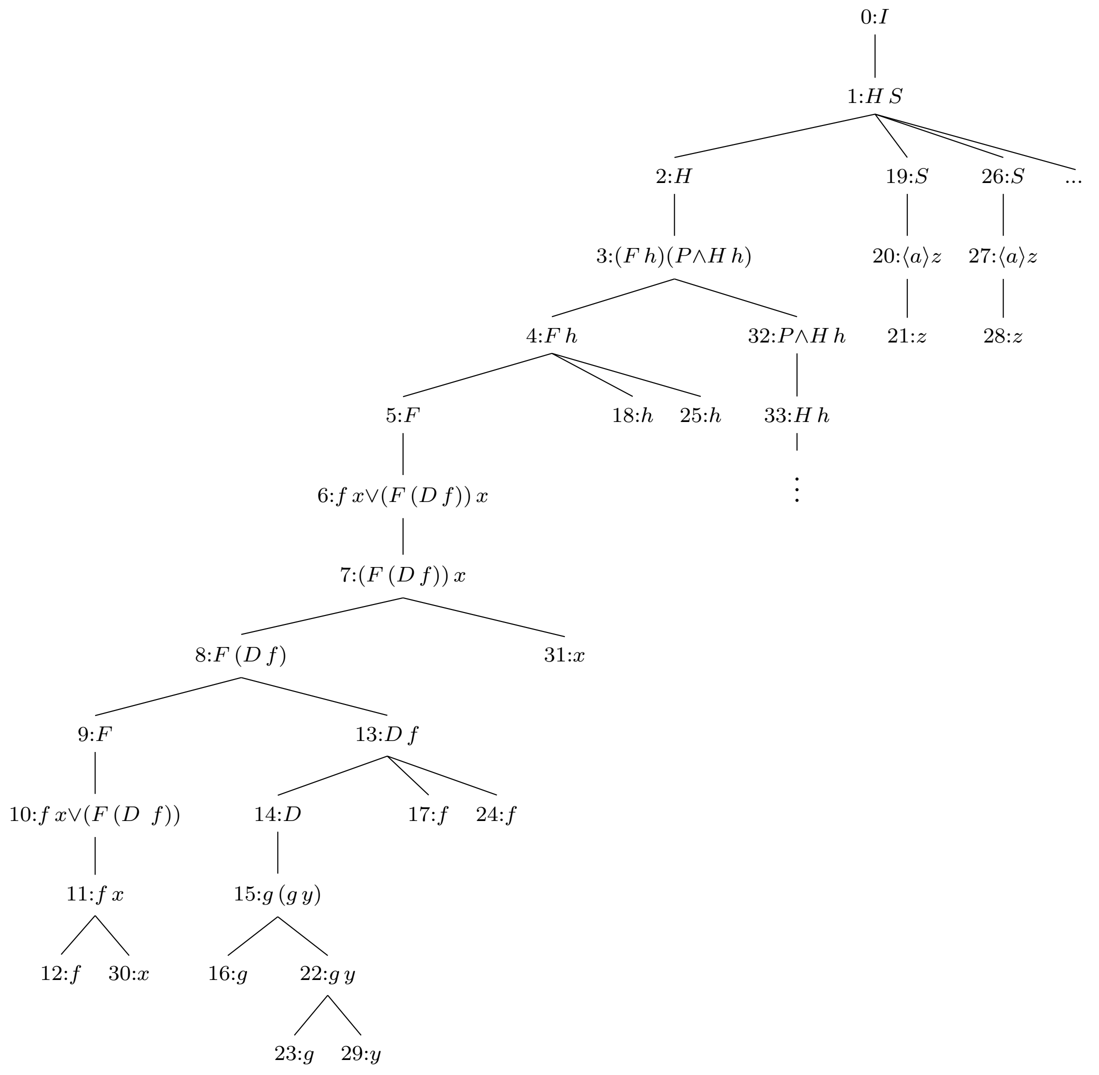
Definition 4.2.15. Let $(C_i)_{i\in\mathbb{N}}$ be a play of the acceptance game of an APKA and let $\mathbb{T}$ be a generalized unfolding tree of that play. Let t be a node in $\mathbb{T}$ labeled by a lambda-variable configuration $C_i = (\_, (f, e), \_, \_)$. We call t *resolved* if the node labeled by C_{i+1} is $t0$, and *unresolved* otherwise. We call $\mathbb{T}$ *resolved down to level k* if all nodes labeled by lambda-variable configurations of type level $k+1$ and above are resolved, and all nodes labeled by lambda-variable configurations of type level k and less are unresolved. We call $\mathbb{T}$ *fully unresolved* or just an *unfolding tree* if all lambda-variable nodes are unresolved.

Since the definition of a generalized unfolding tree together with the requirement of being reduced down to type level k uniquely identify the predecessor of any node in such a tree, for each k there is exactly one such tree.

Observation 4.2.16. Given a play of an acceptance game and $k \geq 0$, its generalized unfolding tree resolved down to type level k is unique.

In light of the previous observation, for a given play π of the acceptance game, we denote with $\mathbb{T}^\pi_k$ its unique associated unfolding tree resolved down to type level k, and with $\mathbb{T}^\pi$ its unique unfolding tree.

Figure 4.3: The upper part of an infinite unfolding tree.



An important property of unfolding trees for infinite plays is that they contain exactly one infinite path, even though they may be infinitely branching. This path contains the infinite recursion responsible for the play being infinite. For example, in the unfolding tree in Figure 4.3, the lower occurrence of H, on the node that is to follow the one labeled 32, is an unfolding of the occurrence of H in the node labeled 2, while the occurrences of F one the left subtree are not going to spawn an infinite recursion.

Theorem 4.2.17. *Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the* APKA *acceptance game and let $\mathbb{T}$ be its associated unfolding tree. Then $\mathbb{T}$ contains exactly one infinite path and there are infinitely many fixpoint configurations on that path.*

The proof is by Lemmas 4.2.32 and 4.2.33 which appear separately in subsection 4.2.6 due to the length of the associated proofs.

4.2.5 The Winner of the Acceptance Game

Theorem 4.2.17 allows us to define the winner of a play of the APKA acceptance game.

Definition 4.2.18. Let $\mathcal{A}$ be an APKA, let $\pi = (C_i)_{i\in\mathbb{N}}$ be an infinite play of its acceptance game on some LTS $\mathcal{T}$ from state v, and let $\mathbb{T}^\pi$ be its associated unfolding tree. Let p be the highest priority of all fixpoint configurations that occur infinitely often on the unique infinite path in $\mathbb{T}$. Then $\mathcal{V}$ wins the play if p is even and $\mathcal{S}$ wins if p is odd. We say that $\mathcal{A}$ accepts the pointed LTS $\mathcal{T}, v$ if and only if $\mathcal{V}$ has a winning strategy in the acceptance game from v. In this case, we write $\mathcal{T}, v \models \mathcal{A}$, and we write $\mathcal{T}, v \not\models \mathcal{A}$ if $\mathcal{S}$ has a winning strategy in the acceptance game from v.

Note that Definition 4.2.18 refers to winning strategies of the players. It is not obvious that, given a pointed LTS and an APKA, there is always exactly one of the players that has a winning strategy. However, as a consequence of APKA capturing HFL (cf. Thm 4.2.19), for each APKA there is an equivalent HFL formula. In fact, $\mathcal{V}$ can derive a winning strategy for the APKA acceptance game from her strategy in the HFL model-checking game for that formula if it holds on a given pointed LTS, and similarly for $\mathcal{S}$. See Section 4.4 for details. Hence, determinacy of the APKA acceptance game follows.

We write $\mathcal{A}_1 \equiv \mathcal{A}_2$ for APKA $\mathcal{A}_1, \mathcal{A}_2$ if for all LTS $\mathcal{T}$ and all $v \in \mathcal{T}$, we have that $\mathcal{T}, v \models \mathcal{A}_1$ if and only if $\mathcal{T}, v, \models \mathcal{A}_2$. Note that an APKA of order 0 is a PA and that the acceptance condition for PA is subsumed by that for APKA.

Theorem 4.2.19. APKA *capture* HFL *in the sense that, for each* HFL *formula* φ *of order* k*, there is an* APKA $\mathcal{A}_\varphi$ *of order* k *such that, for each pointed LTS* $\mathcal{T}, v_0$*, we have that* $\mathcal{T}, v_0 \models \varphi$ *if and only if* $\mathcal{T}, v_0 \models \mathcal{A}_\varphi$ *and, conversely, for each* APKA $\mathcal{A}$ *of order* k *there is an* HFL *formula* $\varphi_\mathcal{A}$ *such that, for each LTS* $\mathcal{T}, v_0$ *we have that* $\mathcal{T}, v_0 \models \mathcal{A}$ *if and only if* $\mathcal{T}, v_0 \models \varphi_\mathcal{A}$.

This is the content of Theorems 4.3.11 and 4.4.9 which can be found in Sections 4.3 and 4.4, respectively.

Observation 4.2.20. From the semantics of the APKA acceptance game it is clear that APKA are invariant under consistent renaming of their states and lambda variables. Moreover, every APKA with at most n priorities is equivalent to one with with priorities in $\{1, \dots, n\}$, respectively $\{0, \dots, n-1\}$.

As is typical for alternating automata, it is rather straightforward to complement a given APKA.

Definition 4.2.21. Let $\mathcal{A} = (\mathcal{Q}, \Delta, Q_I, \delta, (\tau_Q)_{Q\in\mathcal{Q}})$ be an APKA such that the lambda variables of its fixpoint states are exactly those in $\mathcal{F}$. Let $\overline{\mathcal{Q}} = \{\overline{Q} \mid Q \in \mathcal{Q}\}$ and let $\overline{\mathcal{F}} = \{\overline{f}^{\overline{Q}} \mid f^Q \in \mathcal{F}\}$, let $\overline{\Delta}$ be defined via $\overline{\Delta}(\overline{Q}) = \Delta(Q) + 1$, let $\overline{\delta}$ be defined via $\overline{\delta}(\overline{Q}) = \overline{\delta(Q)}^\delta$ which is defined inductively via

$$
\begin{aligned}
\overline{P}^\delta &= \overline{P} \\
\overline{(\overline{P})}^\delta &= P \\
\overline{\chi_1 \vee \chi_2}^\delta &= \overline{\chi_1}^\delta \wedge \overline{\chi_2}^\delta \\
\overline{\chi_1 \wedge \chi_2}^\delta &= \overline{\chi_1}^\delta \vee \overline{\chi_2}^\delta \\
\overline{\langle a \rangle \chi}^\delta &= [a]\overline{\chi}^\delta \\
\overline{[a]\chi}^\delta &= \langle a \rangle \overline{\chi}^\delta
\end{aligned}
$$

$$\overline{\chi_1\,\chi_2}^\delta = \overline{\chi_1}^\delta\,\overline{\chi_2}^\delta$$
$$\overline{f_i^Q}^\delta = \overline{f}_i^{\overline{Q}}$$
$$\overline{Q}^\delta = \overline{Q}$$

and, finally, let $\tau_{\overline{Q}} = \tau_Q$ for $Q \in \mathcal{Q}$. Then $\overline{\mathcal{A}} = (\overline{\mathcal{Q}}, \overline{\Delta}, \overline{Q_I}, \overline{\delta}, (\tau_{\overline{Q}})_{\overline{Q}\in\overline{\mathcal{Q}}})$ is an APKA of the same order and index as $\mathcal{A}$.

Lemma 4.2.22. *Let $\mathcal{A}$ be an* APKA *and let $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ be an LTS of matching vocabulary with distinguished vertex v_I. Then $\mathcal{T}, v_I \models \mathcal{A}$ if and only if $\mathcal{T}, v_I \not\models \overline{\mathcal{A}}$.*

Proof. A player can generate a winning strategy for the acceptance game of $\overline{\mathcal{A}}$ from a winning strategy of the opposing player for $\mathcal{A}$. Following such a strategy generates a play that is similar to one of $\mathcal{A}$, except that all modal and boolean operators have reversed polarity, and the priority of all occurrences of fixpoint states is increased by one with respect to their un-complemented counterparts. In particular, this holds on the infinite path of the unfolding tree of the play, which yields that this strategy is winning for the player if and only if it is winning for the opposing player in the game for $\mathcal{A}$. □

4.2.6 Proof of Theorem 4.2.17

We are now going to prove Theorem 4.2.17, i.e., that any infinite play π of the APKA acceptance game generates an unfolding tree that contains exactly one infinite path. Note that any unfolding tree is a $\mathbb{T}_k^\pi$ for some k bounded by the order of the APKA that generated it. Starting with the full unfolding tree, we show that such a $\mathbb{T}_k^\pi$, with $k \geq 1$, contains an infinite path if and only if $\mathbb{T}_{k-1}^\pi$ contains one. We then show that $\mathbb{T}_0^\pi$ has at most binary branching, which yields, by Kőnig's Lemma, the existence of an infinite path in the tree and, hence all associated generalized unfolding trees. A similar argument shows that the same strategy also transports the existence of several paths. Since we also can show that $\mathbb{T}_0^\pi$ only contains one such infinite path, this settles the argument. It then remains to argue why there are infinitely many fixpoint configurations on the infinite path whose existence we just showed.

For the remainder of this section,we introduce the following notation:

Definition 4.2.23. Let $\pi = (C_i)_{i\in\mathbb{N}}$ be an infinite play of the APKA acceptance game. Let C be a configuration with index i in this play. Then $\mathsf{n}(C)$ denotes C_{i+1} and, if $i > 0$, then $\mathsf{p}(C)$ denotes C_{i-1}. Moreover, $\mathsf{node}_k(C_i)$ denotes the node labeled by C_i in $\mathbb{T}_k^\pi$, and $\mathsf{node}(C)$ denotes the node labeled by C in $\mathbb{T}^\pi$. We write $C \prec_k C'$ if $\mathsf{node}_k(C)$ is an ancestor of $\mathsf{node}_k(C')$ in $\mathbb{T}_k^\pi$ and we write $C \prec C'$ if $\mathsf{node}(C)$ is an ancestor of $\mathsf{node}(C')$ in $\mathbb{T}^\pi$. Finally, if C is an application configuration, we write $C \prec_k^l C'$ to denote that $\mathsf{node}_k(C')$ is in the leftmost subtree below $\mathsf{node}_k(C)$, i.e., operator subtree, and we write $C \prec_k^r C'$ to denote that $\mathsf{node}_k(C')$ is in a non-leftmost below $\mathsf{node}_k(C)$, i.e., an operand subtree.

As a first step, we establish the following: In an unfolding tree, all nodes that are labeled by configurations with closure in some environment e are descendants of the node labeled by the configuration where this environment was created, and, hence, also the node labeled by the associated fixpoint configuration. In fact, later we prove that this is true for all generalized unfolding trees reduced down to any type level.

Lemma 4.2.24. *Consider the unfolding tree $\mathbb{T}^\pi$ of some infinite play $\pi = (C_i)_{i \in \mathbb{N}}$. Let $C = (\_, (\delta(Q), e), \_)$ be the first configuration whose closure component is in environment e, i.e., $\mathsf{p}(C) = (\_, (Q, \_), \_)$ is the configuration where e was created. Let C' be a configuration whose closure component is in environment e. Then $\mathsf{node}(C) \prec \mathsf{node}(C')$, and all nodes on the path from $\mathsf{node}(C)$ to $\mathsf{node}(C')$ are also labeled by configurations whose closure components are in environment e.*

Proof. Consider the predecessor of $\mathsf{node}(C')$ in $\mathbb{T}^\pi$. If it is labeled by $\mathsf{p}(C')$, then by the construction of unfolding trees, the closure component of this configuration is either in e, or it is a fixpoint configuration. In the latter case this fixpoint configuration must be $\mathsf{p}(C)$, since this is the unique fixpoint configuration that is followed by a configuration whose closure component is in e. It follows that $C = C'$.

If the predecessor of $\mathsf{node}(C')$ is not labeled by $\mathsf{p}(C')$, then, by the construction of unfolding trees, $\mathsf{node}(C')$ must be the root of an operand subtree, and $\mathsf{p}(C')$ is of the form $(\_, (f, e'), \_, \_)$. Let C'' be the configuration that labels the predecessor of $\mathsf{node}(C')$ in $\mathbb{T}^\pi$. Then $C'' = \mathsf{bnode}_{e'}(f)$ and, by Lemma 4.2.12, we have that the closure component of this configuration is in environment e, i.e., it is of the form $(\_, (\_, e), \_)$.

By repeated application of the above reasoning, we obtain that the sequence of ancestors of $\mathsf{node}(C'')$ is labeled by configurations whose closure components are in e, until one of the ancestors is labeled by C. Since the closure component of C_0 is in $e^0 \neq e$, such a node labeled by C must eventually appear in the sequence of ancestors of $\mathsf{node}(C')$. This finishes the proof. □

We noted in Lemma 4.2.8 that lambda-variable configurations of ground type behave differently from other lambda variable configurations in the sense that reaching such a configuration means that no later configuration will have a closure in the environment e that defines the ground-type variable. In the context of an unfolding tree, this means that branching at application configurations such that the operand is of ground type is at most binary.

Lemma 4.2.25. *Let $\mathbb{T}^\pi_k$ be a generalized unfolding tree associated to some play π of the* APKA *acceptance game. Let $C = (\_, (\chi\, \chi', \_), \_)$ be an application configuration such that χ' has type $\bullet$. Then $\mathsf{node}_k(C)$ has at most one operand branch.*

Proof. Since C is an application configuration, we know that $C = \mathsf{bnode}_f(e)$ for some e and some lambda variable f of ground type. If $\mathsf{node}_k(C)$ has no operand subtrees, we are done. Otherwise, let C' be the configuration with lowest index such that $\mathsf{node}_k(C')$ is root of an operand subtree of $\mathsf{node}_k(C)$. Since $C = \mathsf{bnode}_f(e)$, we know that $\mathsf{p}(C')$ has closure (f, e). By Lemma 4.2.8, we know that C' is the configuration with the highest index that has a closure in e. It follows that there is no other occurrence of a configuration with closure (f, e) and, hence, no second operand subtree of $\mathsf{node}_k(C)$. □

Our aim is now to prove that the unfolding tree of any infinite play of the acceptance game has exactly one infinite path, since the acceptance condition for APKA depends on that infinite path. For APKA of type level at most 1, this follows from Lemma 4.2.25. Since by the lemma, the tree is of branching degree at most

2, we can use Kőnig's Lemma to obtain an infinite path in the tree. Considerations made formal in Lemma 4.2.32 also yield that there is at most one infinite path.

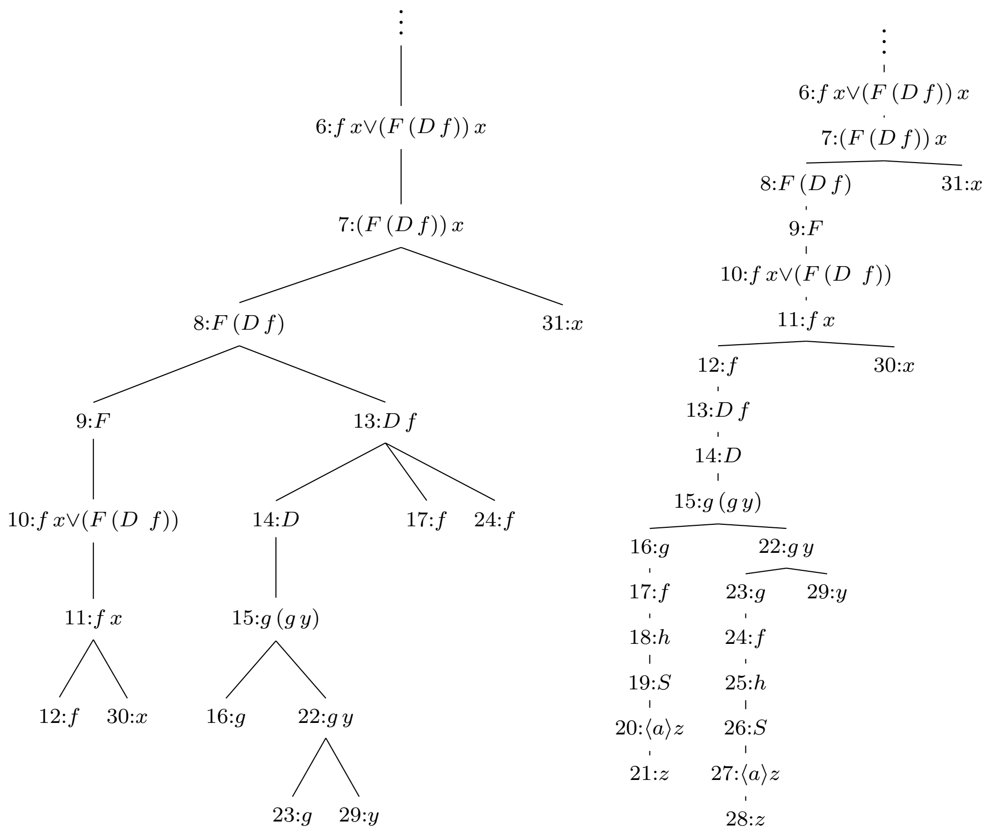
However, for APKA of order more than 1, the situation is more complicated. Since here, the unfolding tree is potentially infinitely branching, we cannot use Kőnig's Lemma to prove the existence of an infinite path. Neither is it straightforward to prove that at most one such path exists. Instead, we will show that, for each infinite play π, and for each $k \geq 0$, the tree $\mathbb{T}^\pi_{k+1}$ has exactly one infinite path if and only if $\mathbb{T}^\pi_k$ has exactly one such path. We show this by reorganizing $\mathbb{T}^\pi_{k+1}$ into $\mathbb{T}^\pi_k$. This means that if a configuration C labels the root of an operand subtree of an application node C' in $\mathbb{T}^\pi_{k+1}$, and if the order of the operand is $k+1$, then $C' = \mathsf{bnode}_f(e)$ for some lambda variable f of order $k+1$ and some environment e, and that $\mathsf{p}(C)$ is of the form $(\_, (f, e), \_)$. In $\mathbb{T}^\pi_k$, we have that $\mathsf{node}_k(C)$ is the son of $\mathsf{node}_k(\mathsf{p}(C))$, i.e., the whole subtree whose root is labeled by C is moved from $\mathsf{node}_{k+1}(C')$ to below the leaf node labeled by the lambda variable configuration that precedes C in π. For example, consider the two partially drawn unfolding trees in Figure 4.4: The left tree is a part of $\mathbb{T}^\pi_1$ for the run in Example 4.1.2; more of the tree is shown in Figure 4.3. The tree on the right is a part of $\mathbb{T}^\pi_0$ of the same play, and with a root labeled by the same configuration. However, note that, for all lambda-variable configurations of order 1 and higher, i.e., f, g and h, the nodes labeled by these configurations are resolved on the right, i.e., the subtree labeled by the following configuration in the play has been moved below the lambda variable node in question. Note that the nodes labeled by configurations $18, \ldots, 21$ and $25, \ldots, 28$ are from parts of $\mathbb{T}^\pi_1$ that are not drawn on the left side. See Figure 4.3 for the missing parts.

This reorganization resembles replacing a formal parameter of a function by its actual value during β-reduction. In fact, the whole reorganization process is quite similar to a normalization argument for the Simply-Typed Lambda Calculus, and in some sense this is exactly what is happening. After the reorganization, the unfolding tree has a shape such that it is easy to see that the reason for the play in question being infinite is not some effect of the higher-order features of HFL, but rather it is due to infinite recursion happening. Moreover, this infinite recursion is then easy to identify due to being on the unique infinite path of the resulting generalized unfolding tree.

A crucial property of $\mathbb{T}^\pi_{k+1}$ is the following: If C is an application configuration such that the operand of the application has order $k+1$, and C' labels the root of an operand subtree of $\mathsf{node}_{k+1}(C)$, then $\mathsf{node}_{k+1}(\mathsf{p}(C'))$, i.e., the node labeled by the variable configuration that resolves to C', is in the operator side subtree of $\mathsf{node}_{k+1}(C)$. Hence, upon passing from $\mathbb{T}^\pi_{k+1}$ to $\mathbb{T}^\pi_k$, the following holds: For all configurations C_1, C_2, if $\mathsf{node}_{k+1}(C_1)$ is an ancestor of $\mathsf{node}_{k+1}(C_2)$ in $\mathbb{T}^\pi_{k+1}$, then $\mathsf{node}_k(C_1)$ is an ancestor of $\mathsf{node}_k(C_2)$ in $\mathbb{T}^\pi_k$ as well. For example, consider again Figure 4.4. In the left tree, we see that $\mathsf{node}_1(C_{13})$ is the right son of $\mathsf{node}_1(C_8)$, and $\mathsf{node}_=(\mathsf{p}(C_{13})1)\mathsf{node}_1(C_{12})$ is in the operator subtree of $\mathsf{node}_1(C_8)$. Moreover, all of these are descendants of $\mathsf{node}_1(C_7)$, and this still holds for the respective nodes in $\mathbb{T}^\pi_0$ on the right.

Contrary to unfolding trees, which are fully unresolved, passing to a tree that is resolved down to some type level breaks the property shown in Lemma 4.2.24. If t is labeled by the configuration where an environment e was created, then in an unfolding tree, for all nodes labeled by configurations whose closure component is in

Figure 4.4: Parts of $\mathbb{T}_2^\pi$ and $\mathbb{T}_1^\pi$ for the play from Example 4.1.2.



e, the path from this node to t is labeled exclusively by configurations whose closure component is in e. Passing to a tree that is resolved down to some type level breaks this property, since configurations whose closure component is in e, who are of a type of order $k+1$, and who label an operand subtree in $\mathbb{T}_{k+1}^\pi$, now label the son of a node labeled by a lambda variable configuration in some other environment. This can be seen in Figure 4.4 as well, even though the environments are not displayed: C_{13} has a closure component in some environment e that was created in C_6 (cf. Figure 4.3). However, $\mathsf{node}_1(C_{13})$ is a son of $\mathsf{node}_1(C_{12})$, which has a closure component in a different environment, namely that created in C_9.

Crucially, however, passing from $\mathbb{T}_{k+1}^\pi$ to $\mathbb{T}_k^\pi$ retains the property that any node labeled by a configuration whose closure component is in e is a descendant of the node labeled by the configuration where e was created. For example, $\mathsf{node}_1(C_{13})$ is still a descendant of $\mathsf{node}_1(C_6)$. Moreover, any lambda variable configuration of the form $(\_, (f, e), \_)$ with f of a type of order $k' \leq k$, labels a node that is an operator branch descendant of the node labeled by $\mathsf{bnode}_f(e)$ in $\mathbb{T}_{k'}^\pi$ due to this reorganization process. This is not necessarily the case in $\mathbb{T}_{k+1}^\pi$, but is our induction invariant for the formal proof of the above.

Lemma 4.2.26. *Let $k \geq 1$ and let $\pi = (C_i)_{i \in \mathbb{N}}$ be an infinite play of the* APKA *acceptance game. Let $C = (\_, (f_i^Q, e), \_)$ be such that f_i^Q is of a type of order k. Let $C' = (\_, (\delta(Q), e), \_)$ be the first configuration with closure component in e. Then $\mathsf{bnode}_e(f_i^Q) \prec_k^l C'$ holds. In particular, if $C' \prec_k C$ holds, then also $\mathsf{bnode}_e(f_i^Q) \prec_k^l C$ holds.*

Proof. Note that, since Q must have parameters, we have that $Q \neq Q_I$ and, hence,

$\mathsf{p}(C')$ is not the first configuration of the play. Let τ be the type of f_i^Q. Since f_i^Q is an argument to Q, the type of Q must have the form $\tau' = \tau_1 \to \cdots \to \tau \to \cdots \to \tau_{k_Q} \to \bullet$. In particular, $ord(\tau') > ord(\tau)$. Now consider the sequence of configurations $C_l = \mathsf{bnode}_e(f_i^Q), \ldots, \mathsf{p}(C') = C_{l+n}$ for some $n \geq 1$. Since by definition of $\mathsf{bnode}_e(f_i^Q)$, the closure $\mathsf{lookup}(f_i^Q, e)$ is on the argument stack for all these configurations except $\mathsf{bnode}_e(f_i^Q)$ itself, the type order for all of these configurations is strictly greater than k and none of them is a fixpoint configuration. Since all these configurations are not of ground type, they all are either application configurations or lambda variable configurations. $\mathsf{bnode}_e(f_i^Q)$ itself is an application configuration.

Let $C_{l+n'}$ be a configuration from the above sequence with $n' > 0$, i.e., $C_{l+n'} \neq \mathsf{bnode}_e(f_i^Q)$. If $C_{l+n'}$ is an application configuration, by the definition of generalized unfolding trees, $\mathsf{node}_k(C_{l+n'+1})$ is the operator branch son of $\mathsf{node}_k(C_{l+n'})$. If, on the other hand, $C_{l+n'}$ is a lambda-variable configuration, the associated lambda variable is of type $k{+}1$ or greater. By assumption, $\mathsf{node}_k(C_{l+n'})$ is reduced and $\mathsf{node}_k(C_{l+n'+1})$ is its unique son in $\mathbb{T}_k^\pi$. Since $\mathsf{node}_k(C_{l+n'+1})$ is a descendant of $\mathsf{node}_k(C_{l+n'})$ for all $0 \geq n' > n$, and, by definition, $\mathsf{node}_k(C_{l+n}) = \mathsf{node}_k(\mathsf{p}(C'))$ is the predecessor of $\mathsf{node}_k(C')$ in $\mathbb{T}_k^\pi$, we obtain that $\mathsf{node}_k(C')$, is a descendant of $\mathsf{node}_k(\mathsf{bnode}_e(f_i^Q))$. In particular, it is in the operator branch below $\mathsf{node}_k(\mathsf{bnode}_e(f_i^Q))$. The statement on $\mathsf{node}_k(C)$ then follows trivially. $\square$

Lemma 4.2.27. *Let $\pi = (C_i)_{i\in\mathbb{N}}$ be a play of the* APKA *acceptance game and k' be the smallest integer such that $\mathbb{T}^\pi = \mathbb{T}_{k'}^\pi$, i.e., $\mathbb{T}^\pi$ is already reduced down to type level k' and k' is the highest type level of lambda-variable leafs, i.e., unresolved lambda-variable nodes, in $\mathbb{T}^\pi$. Then the following are true for all $0 \leq k \leq k'$:*

1. *If f is a lambda variable of order k, defined in some environment e, then for any configuration C of the form $(\_, (f, e), \_)$, we have that* $\mathsf{bnode}_e(f) \prec_k^l C$ *holds.*

2. *If $k < k'$, C, C' are configurations, and $C' \prec_{k+1} C$ holds, then also $C' \prec_k C$ holds. In particular, if C' is an application configuration, we have that $C' \prec_{k+1}^l C$ implies $C' \prec_k^l C$.*

The contents of the lemma are our main induction invariant for the proof of Theorem 4.2.17: A lambda-variable configuration C with closure component (f, e) of order k in $\mathbb{T}_k^\pi$ is alyways in the operator subtree below $\mathsf{node}_k(\mathsf{bnode}_e(f))$, and configurations keep their ancestors while passing to trees reduced down to some lower type level in the sense that any configuration labeling a predecessor of $\mathsf{node}_k(C')$ also labels a predecessor of $\mathsf{node}_{k-1}(C')$. Hence, passing from $\mathbb{T}_k^\pi$ to $\mathbb{T}_{k-1}^\pi$ "moves" the whole subtree below the node $\mathsf{node}_k(\mathsf{n}(C))$ to below $\mathsf{node}_{k-1}(C)$, which remains a descendant of $\mathsf{node}_{k-1}(\mathsf{bnode}_e(f))$ also in $\mathbb{T}_{k-1}^\pi$.

Proof. For each $k \leq k'$, we first show Item 2, then Item 1. Note that, for $k = k'$, Item 1 follows from Lemmas 4.2.26 and 4.2.24, and Item 2 is satisfied trivially.

Let $1 \leq k < k'$ and assume that Items 1 and 2 have been shown for all $k < k'' \leq k'$. We show Item 2 by induction over the index of the configurations in π. In particular, we will show that if $\mathsf{node}_{k+1}(C)$ is a *son* of $\mathsf{node}_{k+1}(C')$, then $\mathsf{node}_k(C)$ is a *descendant* of $\mathsf{node}_k(C')$. The full claim of Item 2 then follows by induction over the structure of $\mathbb{T}_{k+1}^\pi$. The extra claim for application configurations follows since nodes labeled by application configurations have the same leftmost son in all generalized unfolding trees of the same play.

We now do the induction mentioned above. For C_0, there is nothing to show since C_0 labels the root of both $\mathbb{T}^\pi_{k+1}$ and $\mathbb{T}^\pi_k$. Now let C be a configuration in π, and assume that we have shown Item 2 for all configurations of smaller index. Consider the configuration C' such that $\mathsf{node}_{k+1}(C)$ is the son of $\mathsf{node}_{k+1}(C')$ in $\mathbb{T}^\pi_{k+1}$. If $C' = \mathsf{p}(C)$ holds, then $\mathsf{node}_k(C)$ is also a son of $\mathsf{node}_k(C')$, since C' has a boolean or modal closure, a fixpoint closure, a lambda variable closure of type order greater than k, or an application closure and C has the matching operator side closure. If $C' \neq \mathsf{p}(C)$, then by the definition of generalized unfolding trees, C must have a closure component that is the operand side of an application. Moreover, $\mathsf{p}(C)$ has a lambda variable closure (f, e), and $\mathsf{node}_{k+1}(\mathsf{p}(C))$ is not reduced. Finally, $C' = \mathsf{bnode}_e(f)$. If the order of f is less than k, then also $\mathsf{node}_k(\mathsf{p}(C))$ is not reduced, and $\mathsf{node}_k(C)$ is also a son of $\mathsf{node}_k(C') = \mathsf{node}_k(\mathsf{bnode}_e(f))$. The remaining case is that f is of type level k, and $\mathsf{node}_k(\mathsf{p}(C))$ is reduced in $\mathbb{T}^\pi_k$, i.e., $\mathsf{node}_k(C)$ is its unique son. Hence, it suffices to show that $C' = \mathsf{bnode}_e(f) \prec_k \mathsf{p}(C)$ holds. However, by Item 1, we have that $C' \prec_{k+1} \mathsf{p}(C)$ holds. Moreover, since $\mathsf{p}(C)$ has a lower index than C, Item 2 is already shown for it. Hence, $C' \prec_k \mathsf{p}(C)$ holds, and so does $C' \prec_k C$.

It remains to show Item 1. Let $C = (\_\,, (f, e), \_)$ be as in the lemma. Let C' be the first configuration with closure in e, i.e., the unique configuration with closure $(\delta(Q), e)$ for some Q. By Lemma 4.2.24, we know that $\mathsf{node}(C)$ is an operator-branch descendant of $\mathsf{node}(C')$ in $\mathbb{T}^\pi$. By repeated application of Item 2, we obtain that this is also true in $\mathbb{T}^\pi_k$. Consider $C'' = \mathsf{bnode}_f(e)$. By Lemma 4.2.26, we have that $\mathsf{node}_k(C')$ is a descendant of $\mathsf{node}_k(C'')$. Hence, $\mathsf{node}_k(C)$ is a descendant of $\mathsf{node}_k(C'') = \mathsf{node}_k(\mathsf{bnode}_e(f))$ as well. □

Lemma 4.2.28. *Let $\pi = (C_i)_{i\in\mathbb{N}}$ be an infinite play of the* APKA *acceptance game, let $k \geq 0$, let C_a be an application configuration in π, and let C_1 and C_2 be two configurations such that $C_a \prec_{k+1} C_i$ for $i \in \{1, 2, \}$ holds. If $C_1 \prec_k C_2$ holds, then $C_a \prec^l_{k+1} C_1$ holds or $\mathsf{node}_{k+1}(C_1)$ and $\mathsf{node}_{k+1}(C_2)$ are in the same subtree below $\mathsf{node}_{k+1}(C_a)$.*

Proof. We first show that if $C_a \prec^l_{k+1} C$ holds for some configuration C, then the father of $\mathsf{node}_k(C)$ in $\mathbb{T}^\pi_k$ is labeled by a configuration C' such that either $C' = C_a$ or also $C_a \prec^l_{k+1} C'$ holds. There are three cases:

- If $C' = C_a$, the proof is finished.
- If $C' \neq C_a$, but $\mathsf{node}_k(C')$ is also the father of $\mathsf{node}_k(C)$, then $C_a \prec^l_k C$ implies $C_a \prec^l_k C'$.
- If $C' \neq C_a$ and $\mathsf{node}_k(C')$ is not the father of $\mathsf{node}_k(C)$, then by the construction of $\mathbb{T}^\pi_{k+1}$, respectively $\mathbb{T}^\pi_k$, we have that C' is an application configuration and that $\mathsf{node}_{k+1}(C)$ is the root of an operand-side subtree of $\mathsf{node}_{k+1}(C')$. By the construction of $\mathbb{T}^\pi_{k+1}$, we have that $\mathsf{p}(C)$ is of the form $(\_\,, (f, e), \_)$ such that $C' = \mathsf{bnode}_e(f)$, and, by the construction of $\mathbb{T}^\pi_k$, the configuration $\mathsf{p}(C)$ labels the father of $\mathsf{node}_k(C)$ in $\mathbb{T}^\pi_k$. By Lemma 4.2.27, we have that $C' \prec^l_{k+1} \mathsf{p}(C)$ and, moreover $C_a \prec^l_{k+1} C'$. This implies $C_a \prec^l_{k+1} \mathsf{p}(C)$.

In fact, by induction over the path from $\mathsf{node}_k(C)$ to $\mathsf{node}_k(C_a)$, we obtain that all configurations that label nodes on this path in $\mathbb{T}^\pi_k$ label operator-branch descendants of $\mathsf{node}_{k+1}(C_a)$ in $\mathbb{T}^\pi_{k+1}$. By applying this observation to C_2, we obtain that $C_a \prec^l_{k+1} C_2$ implies $C_a \prec^l_{k+1} C_1$.

The remaining case is that $C_a \prec^r_{k+1} C_2$ holds, i.e., C_2 is in some operand subtree below $\mathsf{node}_{k+1}(C_a)$ in $\mathbb{T}^\pi_{k+1}$. Let C^h_2 be the configuration that labels the root of that subtree. Then $\mathsf{p}(C^h_2)$ is of the form $(\_, (f, e), \_)$ and $\mathsf{bnode}_e(f) = C_a$. Hence, by Lemma 4.2.27, we have that $C_a \prec^l_{k+1} \mathsf{p}(C^h_2)$. By the same lemma, $\mathsf{p}(C^h_2)$ labels the father of $\mathsf{node}_k(C^h_2)$ in $\mathbb{T}^\pi_k$, whence $\mathsf{p}(C^h_2) \prec_k C_2$. Note that, by the above observation, for all configurations C' labeling nodes strictly between $\mathsf{node}_k(\mathsf{p}(C^h_2))$ and $\mathsf{node}_k(C_a)$, we have that $C_a \prec^l_{k+1} C'$, i.e., C' labels a node in the operator branch below $\mathsf{node}_{k+1}(C_a)$ in $\mathbb{T}^\pi_{k+1}$.

Now consider $\mathsf{node}_k(C_1)$ and its position in $\mathbb{T}^\pi_k$ relative to $\mathsf{node}_k(C^h_2)$. If $C_1 \prec_k \mathsf{p}(C^h_2)$ holds, then $C_a \prec^l_{k+1} C_1$ follows immediately and we are done. Hence, assume that $\mathsf{p}(C^h_2) \prec_k C_1$ holds. Let C^h_1 be the configuration that labels the root of the subtree below $\mathsf{node}_{k+1}(C_1)$ that contains $\mathsf{node}_{k+1}(C_1)$. Again, we obtain that $C_a \prec^l_{k+1} \mathsf{p}(C^h_1)$ and, of course, $\mathsf{p}(C^h_1) \prec_k C^h_1$. Since $C^h_1 \prec_k \mathsf{p}(C^h_2)$ entails $C_a \prec^l_{k+1} C^h_1$, we have that $C^h_1 \prec_k C^h_2$ does not hold, and neither does $C^h_1 \prec_k C^h_2$. By completely symmetric reasoning, als $C^h_2 \prec_k C^h_1$ does not hold. But since $C_a \prec_k C^h_i \prec_k C_1$ holds for $i \in \{1, 2\}$, we obtain that $C^h_1 = C^h_2$, whence C_1 and C_2 label nodes in the same subtree below $\mathsf{node}_{k+1}(C_a)$, which is as desired. □

Lemma 4.2.29. *Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the* APKA *acceptance game and let $k \geq 0$. If $\mathbb{T}^\pi_{k+1}$ contains at least n pairwise eventually disjoint infinite paths, then so does $\mathbb{T}^\pi_k$.*

Proof. Consider an infinite path in $\mathbb{T}^\pi_{k+1}$ and let $(C_{i_j})_{j\in\mathbb{N}}$ be the sequence of configurations that labels the nodes on this path. Since $C_{i_j} \prec_{k+1} C_{i_{j+l}}$ holds for all $l > 1$, we also have that $C_{i_j} \prec_k C_{i_{j+l}}$ by Item 2 of Lemma 4.2.27. Hence, there must be at least one infinite path in $\mathbb{T}^\pi_k$. Moreover, any sequence of configurations labeling an infinite path in $\mathbb{T}^\pi_{k+1}$ is a subsequence of a sequence of configurations labeling an infinite path in $\mathbb{T}^\pi_k$.

It remains to show that there are actually at least n pairwise eventually disjoint paths in $\mathbb{T}_k$. For $n = 1$ there is nothing to prove. Hence, assume that $n > 1$. Consider any two paths in $\mathbb{T}_{k+1}$. Since they are eventually disjoint, there is a configuration with maximal index that labels a node on both paths. Call this configuration C_a. Since generalized unfolding trees branch on nodes labeled by application configurations, C_a is an application configuration, and at least one infinite path continues into an operand branch below $\mathsf{node}_{k+1}(C_a)$. We call this path the operand path. Let C be the configuration that labels the root of the corresponding operand branch below $\mathsf{node}_{k+1}(C_a)$, and let C' be the configuration that labels the root of the branch that contains the rest of the other path. Note this can be the operator branch below $\mathsf{node}_{k+1}(C_a)$, or another operand branch, and note that $C \neq C'$. Let C'' be any configuration on the other path such that $C' \prec_{k+1} C''$. By Lemma 4.2.28, we have that $C \prec_k C''$ does not hold, since C and C'' do not label nodes in the same subtree below $\mathsf{node}_{k+1}(C_a)$, and C is does not label a node in the operator branch below $\mathsf{node}_{k+1}(C_a)$. Since there are infinitely many configurations like C'', the other path is eventually disjoint from the operand path, since all configurations C'' with $C' \prec_{k+1} C''$ are not descendants of $\mathsf{node}_k(C)$, but all configurations C'' with

$C \prec_{k+1} C''$ are descendants of $\mathsf{node}_k(C)$. Since the two paths were arbitrary, this shows that any two sequences of configurations that label eventually disjoint infinite paths in $\mathbb{T}^\pi_{k+1}$ label nodes on eventually disjoint infinite paths in $\mathbb{T}^\pi_k$. □

Lemma 4.2.30. *Let $\pi = (C_i)_{i \in \mathbb{N}}$ be an infinite play of the* APKA *acceptance game and let $k \geq 0$. If $\mathbb{T}^\pi_k$ contains an infinite path, then so does $\mathbb{T}^\pi_{k+1}$.*

Proof. Fix an infinite path in $\mathbb{T}^\pi_k$. We call a configuration C a *path configuration* if $\mathsf{node}_k(C)$ is on this path, and we call a configuration C *good* if, in $\mathbb{T}^\pi_{k+1}$, there are infinitely many nodes below $\mathsf{node}_{k+1}(C)$ that are labeled by path configurations, i.e., by configurations that label a node on the path in $\mathbb{T}^\pi_k$. Obviously, the initial configuration C_0 is good, since it labels the root of either tree.

We now inductively generate an infinite path labeled by good configurations in $\mathbb{T}^\pi_{k+1}$, starting from the root labeled by C_0. Assume that we have found such path of good configurations up to some configuration C, which itself is good. If $\mathsf{node}_{k+1}(C)$ has only one son in $\mathbb{T}^\pi_{k+1}$, that son must also be labeled by a good configuration. Since $\mathbb{T}^\pi_{k+1}$, like all generalized unfolding trees, branches only at nodes labeled by application configurations, this is the only other type of configuration we have to look at.

Let C be a good application configuration. We show that $\mathsf{node}_{k+1}(C)$ has a descendant labeled by a good configuration. Consider the operand subtrees below $\mathsf{node}_{k+1}(C)$. We claim that at most one of them contains nodes labeled by path configurations. Assume that there are distinct subtrees labeled by configurations C_1 and C_2 that contain nodes labeled by path configurations C^p_1 and C^p_2, respectively. Since $C_i \prec_{k+1} C^p_i$ for either $i \in \{1, 2\}$, by Lemma 4.2.27, we obtain that also $C_i \prec_k C^p_i$ for either $i \in \{1, 2\}$. Hence, both C_1 and C_2 are path configurations and, thus, either $C_1 \prec_k C_2$ must hold, or $C_2 \prec_k C_1$ holds. If the former holds, by Lemma 4.2.28, we obtain that C_1 actually labels the root of the operator subtree below $\mathsf{node}_{k+1}(C)$, if the latter holds, we obtain the same for C_2.

It follows that at most one of the operand branches below $\mathsf{node}_{k+1}(C)$ contains nodes labeled by path configurations. Since, by assumption, there are infinitely many nodes labeled by path configurations below $\mathsf{node}_{k+1}(C)$, but at most the operator subtree and one operand subtree containing nodes labeled by path configurations, at least one of these two trees contains infinitely many nodes labeled by path configurations. Hence, the label of the root of this subtree is good. Since we can always extend a finite path in $\mathbb{T}^\pi_{k+1}$ labeled only by good configurations by another node labeled by a good configuration, using Kőnig's Lemma (Theorem 2.1.1), we obtain that $\mathbb{T}^\pi_{k+1}$ contains an infinite path labeled by good configurations. □

Lemma 4.2.31. *Let $\pi = (C_i)_{i \in \mathbb{N}}$ be an infinite play of the* APKA *acceptance game, and let C_i be an application configuration in the play such that the operand of the application is of ground type. If $\mathsf{node}_0(C_i)$ has an operand subtree in $\mathbb{T}^\pi_0$, then the operator subtree below $\mathsf{node}_0(C_i)$ contains only finitely many nodes.*

Proof. Note that, by Lemma 4.2.25, we have that $\mathsf{node}_0(C_i)$ has at most one operand subtree. We claim that no node in the operator subtree of $\mathsf{node}_1(C_i)$ is labeled by a configuration with index greater than i. For the sake of contradiction, assume that there is in fact such a node in the operator subtree of $\mathsf{node}_0(C_i)$. Let C_j be the configuration with the lowest index $j > i$ that labels a node in the operator subtree of $\mathsf{node}_0(C_i)$. We show that there is another such node with index between i and j that is also in the operator subtree of $\mathsf{node}_0(C_i)$.

Let C be the label of the father of $\mathsf{node}_0(C_j)$. Clearly, $\mathsf{node}_0(C_j)$ is the right son of $\mathsf{node}_0(C)$, for otherwise $C = \mathsf{p}(C_j)$, which would contradict that j is the smallest index greater than i of a configuration labeling a node in the operator subtree below $\mathsf{node}_0(C_i)$. Since $\mathsf{node}_0(C_j)$ is the right son of $\mathsf{node}_0(C)$, we have that $\mathsf{p}(C_j)$ is a lambda variable configuration of the form $(\_, (f, e), \_)$, and by Lemma 4.2.26, we have that $\mathsf{node}_0(\mathsf{p}(C_j))$ is in the operator subtree below $\mathsf{node}_0(C)$. But then $\mathsf{p}(C_j)$ has index $j-1$, but also labels a node in the operator subtree of $\mathsf{node}_0(C_i)$, contradicting that j is the smallest such index greater than i. We conclude that, in fact no configuration with index greater than i labels a node in the operator subtree of $\mathsf{node}_0(C_i)$, which therefore must be finite. □

Lemma 4.2.32. *Let $\pi = (C_i)_{i \in \mathbb{N}}$ be an infinite play of the* APKA *acceptance game and let $\mathbb{T}$ be its unfolding tree. Then $\mathbb{T}$ contains exactly one infinite path.*

Proof. Since $\mathbb{T}_0^\pi$ only branches at application configurations where the operand is of ground type, by Lemma 4.2.25, this tree has at most binary branching. Since it contains infinitely many nodes, by Kőnig's Lemma, it contains an infinite path. By Lemma 4.2.31, the operator subtree of any node with branching, i.e., a node that has an operand subtree, is finite and, hence, contains no infinite path. It follows that no two distinct infinite paths can exist in $\mathbb{T}_0^\pi$, since they can never diverge.

Now let k be the smallest integer such that $\mathbb{T}^\pi$ is also a generalized unfolding tree reduced down to type level k. We know that $\mathbb{T}_0^\pi$ contains exactly one infinite path. By repeated application of Lemma 4.2.30, also $\mathbb{T}_1^\pi, \dots, \mathbb{T}_k^\pi = \mathbb{T}^\pi$ contain at least one infinite path. On the other hand, $\mathbb{T}_0^\pi$ contains at most one infinite path. By the converse of Lemma 4.2.29, so do $\mathbb{T}_1^\pi, \dots, \mathbb{T}_k^\pi = \mathbb{T}^\pi$. We conclude that $\mathbb{T}^\pi$ contains exactly one infinite path. □

After we have established that for an infinite play π, there is exactly one infinite path in each of the $\mathbb{T}_k^\pi$, it remains to prove that there are in fact infinitely many fixpoint configurations on this path. Intuitively, this is due to the fact that the sequence of configurations on said path proceeds from a subformula of the transition relation of a fixpoint state to another, strictly smaller subformula of said state, or to the transition relation of another fixpoint state. Since in $\mathbb{T}^\pi$, lambda-variable configurations appear only on leaves, the infinite path contains no lambda variable-configurations. Also, since the transition relation of each fixpoint state is finite, eventually a new fixpoint configuration must appear on the infinite path.

Lemma 4.2.33. *Let $\pi = (C_i)_{i \in \mathbb{N}}$ be an infinite play of the* APKA *acceptance game. Then the unique infinite path in $\mathbb{T}^\pi$ contains infinitely many nodes labeled by fixpoint configurations.*

Proof. Since the root of $\mathbb{T}^\pi$ is on the infinite path and labeled by $(\_, (Q_I, e^0), \_, \_)$, there is at least one fixpoint configuration on the infinite path. Now consider any node on the infinite path labeled by a fixpoint configuration for some fixpoint Q. Then the unique son of that node is labeled by $(\_, (\delta(Q), e), \_, \_)$ and is also on the infinite path. The following nodes on the infinite path are now labeled by configurations such that the closure components of consecutive configurations are of the forms (χ, e), respectively (χ', e) where χ' is a strict subformula of χ in $\delta(Q)$ until a node is labeled by a configuration of the form (Q', e). Clearly, the node labeled by the configuration with closure $(\delta(Q), e)$ is as desired. Assume that we have proved the statement for all nodes between this node and some node t on the infinite path, and that none of the nodes in between are labeled by fixpoint configurations. Then t is labeled by a configuration with closure (ζ, e) by the induction hypothesis. Then there are two cases. The first case comprises the following subcases:

- If ζ is a boolean or modal formula, then the unique son of t is labeled by a configuration with closure (ζ', e), where ζ' is the respective subformula chosen by one of the players.
- If ζ is an application and the infinite path continues on the operator branch, then the next node is labeled by a configuration with closure (ζ', e) where ζ' is the operator of the application.
- If ζ is an application and the infinite path continues through an operand branch, then the next node on it is labeled by a configuration with closure (ζ', e) where ζ' is the operand of ζ.
- Since t is on the infinite path, and variable configurations only occur at leaves in unfolding trees, ζ is not a lambda variable.

In the second case, ζ is a fixpoint variable as desired. However, since $\delta(Q)$ is a finite HFL-formula, consecutive configurations on the infinite path cannot belong to the first case indefinitely, since that would form an infinite properly descending chain of subformulas in a finite formula. Hence, eventually we obtain another fixpoint configuration on the infinite path. By repeating this argument, we obtain an infinite subsequence of configurations on the infinite path that are all labeled by fixpoint configurations. □

Lemmas 4.2.32 and 4.2.33 together form the proof of Theorem 4.2.17.

Remark 4.2.34. Let $\pi = (C_i)_{i \in \mathbb{N}}$ be an infinite play of the APKA acceptance game. Note that Lemma 4.2.32 states that $\mathbb{T}^\pi$, and each of the $\mathbb{T}^\pi_k$, contain exactly one infinite path. In particular, this is true for $\mathbb{T}^\pi_0$. Moreover, by Lemma 4.2.33, there are infinitely many fixpoint configurations on the infinite path in the unfolding tree. Since any node on an infinite path in the unfolding tree is also on the infinite path in $\mathbb{T}^\pi_0$ (cf. the proof of Lemma 4.2.29), all these fixpoint configurations are also on the infinite path in $\mathbb{T}^\pi_0$. The converse, however, is not true: A fixpoint configuration that is on the infinite path in $\mathbb{T}^\pi_0$ is not necessarily on the infinite path in $\mathbb{T}^\pi$. Since Definition 4.2.18 defines the winner of π via the sequence of the fixpoint configurations on the infinite path in the unfolding tree, it might be tempting to look at the sequence of fixpoint configurations that are on the infinite path in $\mathbb{T}^\pi_0$, but by the above considerations, this yields different semantics.

See the unfolding trees in Figure 6.1, used in Examples 6.2.13 and 6.2.14 for an example of unfolding trees where the fixpoint states on the infinite path differ significantly from those on the infinite path of the associated tree reduced down to type level 0. For example, reducing the variable f in the left tree in Figure 6.1 moves the configuration X above it onto the infinite path. This example underlines that, given an infinite play of the APKA acceptance game, it is not correct to just look at the sequence of fixpoint configurations on the associated generalized unfolding tree reduced down to type level 0.

4.3 From HFL to APKA

The objective of this section is to provide one part of the proof of Theorem 4.2.19, i.e., to show that, for every HFL formula, there is an equivalent APKA of the same order. The translation itself is rather straightforward and follows the same principle as the translation from $\mathcal{L}_\mu$ to PA in Section 2.2.5: Starting from an HFL formula in ANF, we associate a fixpoint state to each fixpoint variable, while the transition relation of such a fixpoint state is deduced from the defining formula of the fixpoint variable. The priorities of the fixpoint states are deduced from the syntactic ordering of the fixpoint variables. However, the correctness proof is more involved, as we have to show equivalence between a formula, which has denotational semantics, and an APKA, which has operational semantics. In order to bridge this gap, we make use of the HFL model-checking game from Section 3.2 which provides operational semantics to HFL formulas. $\mathcal{V}$ can then use her winning strategy in the HFL model-checking game to generate a winning strategy for the APKA acceptance game by keeping both games synchronized via a so-called *strategy mapping* which formalizes the synchronization. Duplicating the argument for spoiler is problematic since the semantics of the HFL model-checking game are not symmetric for the two players.[4] Since such an approach would mean needless duplication of the arguments, we make use of the closure of both HFL and APKA under complementation by showing that the translation from HFL formulas to APKA commutes with complementation.

Let φ be a ground-type HFL formula in ANF. Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be its fixpoint variables, let $X \in \mathcal{X}$ be of type $\tau^X = \tau_1^X \to \cdots \to \tau_{k_X}^X \to \bullet$, and let φ_X be the defining formula of the fixpoint X of the form

$$\sigma_X(X : \tau^X).\, \lambda(x_1^X : \tau_1^X).\, \ldots \lambda(x_{k_X}^X : \tau_{k_X}^X).\, \psi_X$$

where $\sigma_X \in \{\mu, \nu\}$. Let X_I be the top fixpoint variable. Note that since φ is in ANF, we have that $\varphi = \sigma_{X_I} X_I . \psi_{X_I}$.

Let $\mathcal{Q} = \{Q_X \mid X \in \mathcal{X}\}$ be a set of fixpoint states and define a set of lambda variables $\mathcal{F}$ as $\bigcup_{X \in \mathcal{X}} \{f_1^X, \ldots, f_{k_X}^X\}$ with $(f_i^X : \tau_i^X)$ for $X \in \mathcal{X}$ and $1 \leq i \leq k_X$.

Definition 4.3.1. Define $\mathcal{A}_\varphi$ as $(\mathcal{Q}, \Delta, Q_{X_I}, \delta, (\tau_{Q_X} \mid X \in \mathcal{X})$ where $\tau_{Q_X} = \tau_X$ and δ is defined via

$$\delta(Q_X) = \psi_X[\overline{P}/\neg P \mid P \in \mathcal{P}][f_i^X/x_i^X \mid 1 \leq i \leq k_X][Q_Y/Y \mid Y \in \mathcal{X}][Q_Y/\varphi_Y \mid Y \in \mathcal{X}]$$

i.e., $\delta(Q_X) = \mathsf{to}_\delta(\psi_X)$ where

$$\begin{aligned}
\mathsf{to}_\delta(P) &= P \\
\mathsf{to}_\delta(\neg P) &= \overline{P} \\
\mathsf{to}_\delta(\psi_1 \vee \psi_2) &= \mathsf{to}_\delta(\psi_1) \vee \mathsf{to}_\delta(\psi_2) \\
\mathsf{to}_\delta(\psi_1 \wedge \psi_2) &= \mathsf{to}_\delta(\psi_1) \wedge \mathsf{to}_\delta(\psi_2) \\
\mathsf{to}_\delta(\langle a \rangle \psi') &= \langle a \rangle \mathsf{to}_\delta(\psi') \\
\mathsf{to}_\delta([a]\psi') &= [a]\mathsf{to}_\delta(\psi') \\
\mathsf{to}_\delta(\psi_1\, \psi_2) &= \mathsf{to}_\delta(\psi_1)\, \mathsf{to}_\delta(\psi_2) \\
\mathsf{to}_\delta(x_i^X) &= f_i^X \\
\mathsf{to}_\delta(Y) &= Q_Y \\
\mathsf{to}_\delta(\sigma Y.\, \psi') &= Q_Y
\end{aligned}$$

[4]This is intended but could be changed.

and Δ is defined inductively along $\succ$ via

$$\begin{array}{ll}
\Delta(Q_X) = 0 & \text{if } \{Y \mid X = \mathit{upVar}(Y)\} = \emptyset, \sigma_X = \nu \\
\Delta(Q_X) = 1 & \text{if } \{Y \mid X = \mathit{upVar}(Y)\} = \emptyset, \sigma_X = \mu \\
\Delta(Q_X) = \max\{\Delta(Q_Y) \mid X = \mathit{upVar}(Y)\} & \text{if } \sigma_Y = \sigma_X \text{ f.a. } Y \\
 & \text{s.t. } X = \mathit{upVar}(Y) \\
\Delta(Q_X) = 1 + \max\{\Delta(Q_Y) \mid X = \mathit{upVar}(Y)\} & \text{if ex. } Y \text{ s.t. } X = \mathit{upVar}(Y) \\
 & \text{and } \sigma_X \neq \sigma_Y
\end{array}$$

Note that if $X \succ Y$ then $\Delta(Q_X) \geq \Delta(Q_Y)$.

Lemma 4.3.2. *$\mathcal{A}_\varphi$ is well defined and of the same order as φ.*

Proof. The claim on the orders follows from the fact that $\tau_{Q_X} = \tau_X$. Let Σ^X contain, for each $X \in \mathcal{X}$, the hypothesis $Q_X \colon \tau_X$ and the hypotheses $f_1^X : \tau_1^X, \dots, f_{k_X}^X : \tau_{k_X}^x$. Note that this is a set of of APKA typing hypotheses, even though the indices stem from φ. Well-definedness of $\mathcal{A}_\varphi$ entails verification that, for each X and, hence for each Q_X, the judgment $\Sigma^X \vdash \delta(Q_X) \colon \bullet$ is derivable. Such a derivation can be obtained from the derivation of $\psi_X \colon \bullet$ by replacing the HFL assumptions by their counterparts in $\mathcal{A}_\varphi$ and replacing the derivation for φ_Y with $X = \mathit{upVar}(Y)$ by the axiom for $Q_Y \colon \tau_Y$. $\square$

4.3.1 The Correctness Proof

In order to prove the equivalence of φ and $\mathcal{A}_\varphi$, we use the HFL model-checking game defined in Section 3.2. Given an LTS $\mathcal{T}$ such that $\mathcal{T}, v_I \models \varphi$, player $\mathcal{V}$ will use her winning strategy in the model-checking game for $\mathcal{T}, v_I$ and φ to construct an accepting run of $\mathcal{A}_\varphi$ starting from $\mathcal{T}, v_I$. This is done by keeping the current position in the model-checking game similar to the current configuration in the acceptance game. If $\mathcal{V}$ has to make a decision in the acceptance game, she consults her strategy in the model-checking game in order to obtain a good choice. If $\mathcal{S}$ is to make a decision in the acceptance game, $\mathcal{V}$ mimics this move in the model-checking game in order to keep the positions synchronized. In order to make this notion formal, we employ a function expand that replaces lambda variables in a closure by the result of variable lookup in question.

Definition 4.3.3. Let (χ, e) be a closure in a play of the acceptance game for some APKA. Define the function expand[5] via:

$$\begin{aligned}
&\mathsf{expand}(\chi, e) = (\chi, e) \text{ if } \chi \text{ is not a lambda variable} \\
&\mathsf{expand}(f, e) = \mathsf{expand}(\mathsf{lookup}(f, e)) \text{ if } f \text{ is a lambda variable}
\end{aligned}$$

Given an HFL formula in the context of an HFL model-checking game and a closure in the context of an APKA acceptance game, a *strategy mapping* is a function that associates to each node in the syntax tree of the formula a closure such that

- the top operators of formula and closure agree and

[5]In order to avoid double parentheses, we display $\mathsf{expand}(c)$ as $\mathsf{expand}(\chi, e)$ instead of $\mathsf{expand}((\chi, e))$.

- the mapping is compatible with the structure of the formula, e.g., the mapping of a conjunction is the conjunction of the mappings of the conjuncts.

Note that the formulas that occur in the HFL model-checking game do not have free lambda variables and also do not contain lambda abstractions. Formally, a strategy mapping is defined as follows:

Definition 4.3.4. Let ψ be a lambda-variable-closed HFL formula that contains no lambda abstractions and let c be a well-defined closure in $\mathit{Clos}(\mathcal{A})$ for some APKA $\mathcal{A}$ with fixpoint states $\mathcal{Q}$. We say that m is a *strategy mapping* from the syntax tree $\mathbb{T}_\psi$ to c if $m(\varepsilon) = c$ and for all nodes $t \in \mathbb{T}_\psi$ the following holds:

- If t is labeled by P or $\neg P$, then $m(t) = (P, \_)$, respectively $m(t) = (\overline{P}, \_)$
- If t is labeled by $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$, then $m(t) = (\chi_1 \vee \chi_2, e)$, respectively $(\chi_1 \wedge \chi_2, e)$, and $m(t1) = \mathsf{expand}(\chi_1, e)$ and $m(t2) = \mathsf{expand}(\chi_2, e)$.
- If t is labeled by $\langle a \rangle \psi'$ or $[a]\psi'$, then $m(t) = (\langle a \rangle \chi', e)$, respectively $m(t) = ([a]\chi', e)$ and $m(t1) = \mathsf{expand}(\chi', e)$.
- If t is labeled by $\psi_1\, \psi_2$ then $m(t) = (\chi_1\, \chi_2, e)$ and $m(t1) = \mathsf{expand}(\chi_1, e)$ and $m(t2) = \mathsf{expand}(\chi_2, e)$.
- If t is labeled by X^s then $m(t) = (Q, e')$ for some $Q \in \mathcal{Q}$.

We say that two strategy maps m_1, m_2 *agree on fixpoint variables* if, given two nodes t, u in the domains of m_1, respectively m_2 and labeled by X^s, respectively $Y^{s'}$, we have that $m_1(t) = (Q, e) = m_2(u)$ implies $X = Y$ and $s = s'$.

In effect, a strategy mapping from a suitable HFL formula, usually in the context of an HFL model-checking game, into a well-defined closure that appears in context of an APKA acceptance game ensures that the formula and the substate part of the closure are structurally the same after factoring in variable lookup via expand. Moreover, if two strategy maps agree, a fixpoint closure (Q, e) uniquely determines the fixpoint variable and signature on all of its preimages. We use this to simplify the tracking of said fixpoint variables and their closures throughout a pair of the HFL model-checking game for φ and the acceptance game for $\mathcal{A}_\varphi$. This is particularly useful when making statements on the nature of the variables that occur in conjunction with the acceptance condition of the APKA acceptance game, i.e., the variables that occur on the infinite path of the unfolding tree of the play. This is useful since said infinite path cuts out parts of the play if the path goes through the operand side of an application node. Instead of having to characterize the behavior of fixpoint variables in the part of the play that was cut out, we can characterize the behavior on the APKA side and then rely on the fact that the strategy mappings involved agree on fixpoints. This can be made formal as follows:

Definition 4.3.5. Let $(v, (\chi, e), \Gamma)$ and $(v', \psi, \digamma)$ be configurations in the acceptance game for some APKA, respectively the model-checking game for some HFL formula. Let $c_1, \ldots, c_k$ and $\psi_1, \ldots, \psi_{k'}$ be the contents of Γ and $\digamma$ from top to bottom. We say that the pair is *good* if:

1. $\mathcal{V}$ wins from $(v', \psi, \digamma)$, and

2. $v = v'$, and

3. $k = k'$ and,

4. there are *strategy mappings* $m, m_1, \ldots, m_k$ from the syntax trees of ψ and $\psi_1, \ldots, \psi_k$ to (χ, e), respectively to $c_1, \ldots, c_k$ that agree on fixpoint variables.

Given two configurations C_1 and C_2 of the APKA acceptance game for some APKA, and two positions P_1 and P_2 in the HFL model-checking game for some HFL formula such that C_2 is a valid successor of C_1 and P_2 is a valid successor of P_1, we say that the pair of transitions is *good* if the pair of C_i and P_i is good for $i \in \{1, 2\}$ and if all strategy mappings involved agree on fixpoints.

Note that Definition 4.3.5 is not restricted to the APKA $\mathcal{A}_\varphi$ that was defined in the earlier parts of this section, but can be any APKA. The motivation behind this deviation from the APKA fixed at the beginning of the section is that we will reuse this definition in Section 4.4 which describes the translation of an APKA into an HFL formula. The intuition here is that, if starting from a good pair of positions, and if both games develop such that the involved pairs of positions are good, and also the pairs of transitions are good, then the fact that the preimage of a closure (Q, e) uniquely determines its preimage under a given strategy map generalizes to all pairs of positions involved.

We now show that $\mathcal{V}$ can leverage her strategy in the HFL model-checking game for φ into a strategy in the acceptance game for $\mathcal{A}_\varphi$ such that both games proceed in good transitions from one good pair of positions to the next and such that $\mathcal{V}$ does not lose the acceptance game for $\mathcal{A}_\varphi$ in finitely many steps.

Lemma 4.3.6. *Let φ be an* HFL *formula and let $\mathcal{T}, v_I$ be an LTS such that $\mathcal{T}, v_I \models \varphi$. Then the pair $(v_I, (Q_{X_I}, e^0), \varepsilon)$ and $(v_I, X_I^{s_I}, \varepsilon)$ of starting positions in the acceptance game for $\mathcal{A}_\varphi$ and the* HFL *model-checking game for φ is good. Moreover, for any given good pair C, P of positions in those two games, either $\mathcal{V}$ wins both games immediately, or there are successor positions C' of C, respectively P' of P such that the pair C', P' is also good and, moreover, the transition is good.*

Proof. Since $\mathcal{T}, v_I \models \varphi_{\mathcal{A}}$ we have that $\mathcal{V}$ wins the HFL model-checking game for $\varphi_{\mathcal{A}}$. Moreover, both stacks are empty, so the conditions pertaining to them are satisfied trivially. The mapping m with $m(X_I^{s_I}) = (Q_{X_I}, e^0)$ clearly is a valid strategy mapping. Hence, the pair of starting positions is good.

Now suppose $C = (v, (\chi, e), \Gamma)$ and $P = (v, \psi, \digamma)$ is a good pair of positions. Let $(c_1, \ldots, c_k)$ be the contents of Γ from top to bottom, and let $\psi_1, \ldots, \psi_k$ be the contents of $\digamma$ from top to bottom. Let m and $m_1, \ldots, m_k$ be the strategy mappings for ψ respectively $\psi_1, \ldots \psi_k$. Depending on the form of χ, we show that $\mathcal{V}$ either wins both games immediately, or that there is a good pair of transitions available:

- If $\chi = P$ or $\chi = \overline{P}$, then also $\psi = P$, respectively $\psi = \neg P$ since $m(\psi) = \mathsf{expand}(\chi, e) = (\chi, e)$. Since $\mathcal{V}$ wins in the HFL model-checking game, we have that $\mathcal{T}, v \models \psi$ and, hence, also $\mathcal{T}, v \models \chi$, whence she also wins the acceptance game for $\mathcal{A}_\varphi$.

- If $\chi = \chi_1 \vee \chi_2$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \psi_1 \vee \psi_2$. Since $\mathcal{V}$ has a winning strategy in the HFL model-checking game for φ, she picks

$i \in \{1,2\}$ according to that strategy. The acceptance game for $\mathcal{A}_\varphi$ continues from $C' = (v, (\chi_i, e), \Gamma)$ and the HFL model-checking game for φ continues from $P' = (v, \psi_i, \digamma)$. By assumption, $\mathcal{V}$ also wins from P'. Since both stacks have not been altered, the conditions pertaining to them and their associated strategy mappings are satisfied by assumption, respectively by keeping the strategy mappings from the pair C, P. The strategy mapping m' for ψ_i is obtained as the restriction of m to the subtree induced by the φ_i. Clearly, this is a valid strategy mapping that agrees with m on fixpoint variables, and hence all other strategy mappings involved. Since $m'(t) = m(ti)$ for any node t in the syntax tree of ψ, the pair of transitions is also good.

- If $\chi = \chi_1 \wedge \chi_2$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \psi_1 \wedge \psi_2$. Then $\mathcal{S}$ picks $i \in \{1,2\}$ and the acceptance game for $\mathcal{A}_\varphi$ continues from $C' = (v, (\chi_i, e), \Gamma)$ and the HFL model-checking game for φ continues from $P' = (v, \psi_i, \digamma)$. Since $\mathcal{V}$ has a winning strategy in the HFL model-checking game for $\varphi_\mathcal{A}$, $\mathcal{V}$ also wins from P'. The rest of the argument proceeds as in the previous case.

- If $\chi = \langle a \rangle \chi'$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \langle a \rangle \psi'$. Since $\mathcal{V}$ has a winning strategy in the HFL model-checking game for $\varphi_\mathcal{A}$, there is w such that $v \xrightarrow{a} w$ and $(w, \psi', \digamma)$ is winning for $\mathcal{V}$. The acceptance game for $\mathcal{A}_\varphi$ continues from $C' = (w, (\chi', e), \Gamma)$ and the HFL model-checking game for φ continues from $P' = (w, \psi', \digamma)$. The conditions pertaining to the stack contents are satisfied as in the previous cases. The strategy mapping m' for ψ' is obtained as the restriction of m to the subtree induced by the φ'. By the same reasoning as above, this is a valid strategy mapping that agrees on fixpoints with all strategy mappings involved, whence the pair of transitions is also good.

- If $\chi = [a]\chi'$, from $m(\psi) = (\chi, e)$ it follows that $\psi = [a]\psi'$. Hence, $\mathcal{S}$ picks w such that $v \xrightarrow{a} w$, if possible. If not, $\mathcal{V}$ wins both games immediately. If $\mathcal{S}$ is not stuck, by assumption, $\mathcal{V}$ wins from $(w, \psi', \digamma)$. The rest of the argument proceeds as in the previous case.

- If $\chi = \chi'_1\,\chi'_2$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \psi'_1\,\psi'_2$. The acceptance game for $\mathcal{A}_\varphi$ continues in $C' = (v, (\chi'_1, e), \Gamma')$ and the HFL model-checking game for φ continues from $(v, \psi'_1, \digamma')$ where the contents of Γ are $(\chi'_2, e), c_1, \ldots, c_k$ from top to bottom, and the contents of $\digamma'$ are $\psi'_2, \psi_1, \ldots, \psi_k$ from top to bottom. Since $\mathcal{V}$ wins from P, she also wins from P'. Since the stacks gained one element each, the condition pertaining to their sizes is satisfied in the pair C', P'. For the bottom k elements, the strategy mappings continue to be $m_1, \ldots, m_k$ from top to bottom. The strategy mapping m'_2 from the new topmost stack element ψ'_2 of $\digamma$ to the new topmost element (χ'_2, e) is obtained as the restriction of m to the subtree induced by ψ'_2 in ψ, which is a valid strategy mapping since m is. The strategy mapping m'_1 from ψ'_1 to (χ'_1, e) is obtained as the restriction of m to the subtree induced by ψ'_1 in ψ, which is a valid strategy mapping for the same reason. Since the new strategy mappings are restrictions of strategy mappings from the pair C, P, not only is the pair C', P' good, but the transition is also good.

- If $\chi = f_i^Q$, then the acceptance game for $\mathcal{A}_\varphi$ continues in the configuration $C' = (v, \mathsf{lookup}(f_i^Q, e), \Gamma)$ and the HFL model-checking game for φ remains in

P. By definition, $m(\psi) = \mathsf{expand}(\chi, e) = \mathsf{expand}(\mathsf{lookup}(f_i^Q, e))$, whence m is also a valid strategy map for ψ in the pair C', P. Since everything else stays put, the new pair is good and so is the transition.

- if $\chi = Q$, since $m(\psi) = (\chi, e)$, we have that $\psi = X^s$ for some μ-signature s for X. If X is a least-fixpoint variable, by assumption, $s(X) \neq 0$. This is since otherwise $\mathcal{V}$ loses the HFL model-checking game, contradicting that the pair of positions is good. If X is a least-fixpoint variable and $s(X)$ is a limit ordinal, let $\alpha < s(X)$ be the ordinal that $\mathcal{V}$ picks in the HFL model-checking game. Otherwise, X is a least-fixpoint variable and $s(X) = \alpha + 1$, or X is a greatest-fixpoint variable. Let s' be the new signature defined in the HFL model-checking game, i.e., s' is defined via

  $$\begin{aligned} s'(Y) &= \alpha && \text{if } Y = X \\ s'(Y) &= \mathsf{ht}(\llbracket \tau_Y \rrbracket_{\mathcal{T}}) && \text{if } X = \mathit{upVar}(Y) \text{ and } \sigma_Y = \mu \\ s'(Y) &= s(Y) && \text{otherwise,} \end{aligned}$$

  if X is a least-fixpoint variable, and otherwise as

  $$\begin{aligned} s'(Y) &= \mathsf{ht}(\llbracket \tau_Y \rrbracket_{\mathcal{T}}) && \text{if } X = \mathit{upVar}(Y) \text{ and } \sigma_Y = \mu \\ s'(Y) &= s(Y) && \text{otherwise.} \end{aligned}$$

  Let $\mathcal{X}_1 = \{Y \mid Y \succ X\} \cup \{X\}$, and let $\mathcal{X}_2 = \{Y \mid X = \mathit{upVar}(Y)\}$. The HFL model-checking game continues in the position (v, ψ', ε) where

  $$\psi' = \varphi'_X[\psi_i/x_i^X \mid 1 \leq i \leq k_X][Y^{s'}/Y \mid Y \in \mathcal{X}_1][Y^{s'}/\varphi''_Y \mid Y \in \mathcal{X}_2]$$

  and $\varphi''_Y = \varphi_Y[Z^{s'}/Z \mid Z \in \mathcal{X}_1]$. On the other hand, the acceptance game for $\mathcal{A}$ continues in the configuration $C' = (v, (\delta(Q_X), e'), \varepsilon)$ where $e' = (f_1^X \mapsto c_1, \dots, f_{k_X}^X \mapsto c_k, \_, \_)$ and $\delta(Q_X) = \mathsf{to}_\delta(\psi_X)$.

  We now define the new strategy mapping m' by induction over the syntax tree of ψ'. The initial mapping of m' is $m'(\psi') = \mathsf{expand}(\mathsf{to}_\delta(\psi_X), e')$. Let t be a node in the syntax tree of $\mathsf{to}_\delta(\psi_X)$ of the form

  $$\psi''\underbrace{[\overline{P}/\neg P \mid P \in \mathcal{P}][\psi_i/x_i^X \mid 1 \leq i \leq k][Y^{s'}/Y \mid Y \in \mathcal{X}_1 \cup \mathcal{X}_2]}_{\kappa}$$

  where ψ'' is a subformula of ψ_X. We then write $\psi''[\kappa]$ to denote

  $$\psi''[\overline{P}/\neg P \mid P \in \mathcal{P}][\psi_i/x_i^X \mid 1 \leq i \leq k][Y^{s'}/Y/Y \mid Y \in \mathcal{X}_1 \cup \mathcal{X}_2]$$

  in order to improve readability.

  Depending on the top operator of the formula ψ'' for t, we continue to define m'. In order to improve readability, we will not explicitly display all substitutions:

  - If ψ'' is P or $\neg P$, there is nothing to define.
  - If ψ'' is $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$, then

    $$m'(\psi''[\kappa]) = \mathsf{expand}(\mathsf{to}_\delta(\psi''), e') = (\mathsf{to}_\delta(\psi')', e')$$

    which is $(\mathsf{to}_\delta(\psi_1) \vee \mathsf{to}_\delta(\psi_2), e')$, respectively $(\mathsf{to}_\delta(\psi_1) \wedge \mathsf{to}_\delta(\psi_2), e')$. Define $m'(\psi''_i[\kappa]) = \mathsf{expand}(\mathsf{to}_\delta(\psi''), e')$ for $i \in \{1, 2\}$.

- If ψ'' is $\langle a\rangle\psi'''$ or $[a]\psi'''$ then $m'(\psi''[\kappa]) = \mathsf{expand}(\mathsf{to}_\delta(\psi''), e)$ which is $(\langle a\rangle\mathsf{to}_\delta(\psi'''), e')$, respectively $([a]\mathsf{to}_\delta(\psi'''), e')$. Define

$$m'(\psi'''[\kappa]) = \mathsf{expand}(\mathsf{to}_\delta(\psi'''), e').$$

- If ψ'' is $\psi_1\,\psi_2$ then

$$m'(\psi''[\kappa]) = \mathsf{expand}(\mathsf{to}_\delta(\psi''), e')$$

  which is $(\mathsf{to}_\delta(\psi_1)\,\mathsf{to}_\delta(\psi_2), e')$. Define $m'(\psi_i''[\kappa]) = \mathsf{expand}(\mathsf{to}_\delta(\psi''), e')$ for $i \in \{1, 2\}$.
- If ψ'' is Y then $\psi''[\kappa] = Y^{s'}$ and $m'(\psi''[\kappa]) = (Q_Y, e')$. There is nothing to define, but note that e' is new and, hence, closures of the form $(\_\,, e')$ do not appear in the range of m, respectively $m_1, \ldots, m_k$.
- If ψ'' is $f_i^X[\kappa]$, then $\psi''[\kappa] = \psi_i$, and

$$\mathsf{expand}(\mathsf{to}_\delta(x_i^X), e') = \mathsf{expand}(\mathsf{lookup}(f_i^X, e')) = c_i.$$

  Since m_i is a valid and good strategy mapping from ψ_i to c_i, we can use it do define m' via $m'(tu) = m_i(u)$ if u is in the domain of the syntax tree of ψ_i. Hence, m' is a valid strategy mapping on this subtree.

Clearly, m' is a valid strategy mapping from ψ' to $(\delta(Q_X), e')$. It is also good: On the subtrees of ψ' that are derived from substitution of one of the ψ_i, by definition m' agrees with m_i, which is good by assumption. On the parts of the tree derived from $\delta(Q)$, for all nodes in the syntax tree that are labeled by a fixpoint variable Y, we have that Y is of the form $Y^{s'}$. Hence, two nodes where m' yields (Q_Y, e) must both be labeled by $Y^{s'}$. Moreover, closures of this form do not appear in the range of the m_i, so m' is good. By the same reasoning, the pair of transitions from C to C' and from P to P' is also good.

□

If $\mathcal{V}$ plays according to the strategy leveraged from the HFL model-checking game, by Lemma 4.3.6, she avoids losing in finitely many steps. Before we give the main result, we make sure that the strategy maps involved in the proof behave as expected in the sense that they map subformulas X^s only to closures $(Q_Y, \_)$ such that $Y = X$.

Observation 4.3.7. Let φ be an HFL formula and let $\mathcal{T}, v_I$ be an LTS such that $\mathcal{T}, v_I \models \varphi$. Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the acceptance game for $\mathcal{A}_\varphi$ such that $\mathcal{V}$ plays with her strategy leveraged from φ, and let $(P_i)_{i\in\mathbb{N}}$ be the sequence of positions in the HFL model-checking game associated with the play. Let m be any strategy map in the pair C_i and P_i for some $i \in \mathbb{N}$. If some node t in the domain of m is labeled by X, then $m(X) = (Q_X, \_)$.

This is easy to see by inspecting the case for fixpoint variables in the proof of Lemma 4.3.6 which is the only place where fresh strategy mappings are defined, as opposed to using restrictions of previous mappings. Since all subsequent mappings agree on fixpoints with the new mapping, the result then follows.

Lemma 4.3.8. *Let φ be an* HFL *formula in ANF and let $\mathcal{T}, v_I$ be an LTS such that $\mathcal{T}, v_I \models \varphi$. Let $(C_i)_{i \in \mathbb{N}}$ be a play of the acceptance game for $\mathcal{A}_\varphi$ where $\mathcal{V}$ plays with her strategy leveraged from the model-checking game for φ. Let $C_i = (\_, (Q_X, \_), \_, \_)$ and $C_{i'} = (\_, (Q_Y, e), \_)$ be two distinct fixpoint configurations on the infinite path in the unfolding tree of the play such that $C_{i'}$ is the lower one and no fixpoint configuration appears on the infinite path between C_i and $C_{i'}$. Let P_i and $P_{i'}$ be the positions in the* HFL *model-checking game associated to C_i and $C_{i'}$ by $\mathcal{V}$'s strategy. Then the following is true:*

1. *$P_i = (\_, X_s, \_)$ and $P_{i'} = (\_, Y_{s'}, \_)$,*
2. *s' is descending from s with respect to Y.*
3. *$X = Y$ or X and Y are comparable by $\succ$.*

Proof. Because both pairs of positions are good, and by Observation 4.3.7, we have that $P_i = (\_, X_s, \_)$ and $P_{i'} = (\_, Y_{s'}, \_)$. Note that the strategy mapping m for $P_{i'}$ and $C_{i'}$ maps $Y^{s'}$ to (Q_Y, e').

Let C_{i+1} be the configuration after C_i. By definition of the APKA acceptance game, $C_{i+1} = (\_, (\delta(Q_X), e'), \varepsilon)$ for some environment e'. In fact, $e' = e$. This can be seen by an induction over the path between the node labeled by C_i and the node labeled by $C_{i'}$ since, by the construction of an unfolding tree, any son of a node labeled by $C = (\_, (\chi, e), \_)$ is of the form $C' = (\_, (\zeta, e), \_)$ unless $\chi = Q$ is a fixpoint formula. For leftmost sons, this is immediate from the definition of the transition relation of APKA and the fact that lambda variable configurations occur only at leaves in an unfolding tree. For non-leftmost sons, $\chi = \chi_1\, \chi_2$ is necessarily an application, and $\zeta = \chi_2$ since the configuration immediately preceding C' is of the form $(\_, (f, e'), \_)$ and $C = \mathsf{bnode}_{e'}(f)$. By the same reasoning, we also obtain that ζ is a subformula of χ. Applying this to C_i and $C_{i'}$, we obtain that $e = e'$ and, moreover, that Q_Y is a subformula of $\delta(Q_X)$. This also entails that X and Y are comparable by $\succ$ unless actually $X = Y$ holds.

Now consider the position P_{i+1} associated to C_{i+1} by $\mathcal{V}$'s strategy. It has the form $(\_, \psi', \_)$ such that $Y_{s'}$ occurs in ψ', since there is a strategy mapping m' from ψ' to $(\delta(Q_X), \_)$ and (Q_Y, e') appears in its range. This is because Q_Y occurs as a subformula in $\delta(Q_X)$ and m and m' agree on fixpoint variables. By the definition of the HFL model-checking game, s' is strictly descending from s with with respect Y. □

We now show that $\mathcal{V}$ not only can avoid losing any finite play of the APKA acceptance game for $\mathcal{A}_\varphi$ if she uses her strategy leveraged from the model-checking game for φ, but that she actually wins any infinite play of the APKA acceptance game when using this strategy.

Lemma 4.3.9. *Let φ be an* HFL *formula in ANF and let $\mathcal{T}, v_I$ be an LTS such that $\mathcal{T}, v_I \models \varphi$. Then $\mathcal{T}, v_I \models \mathcal{A}_\varphi$.*

Proof. Assume that $\mathcal{V}$ plays with her strategy leveraged from the model-checking game for φ. By Lemma 4.3.6, she wins every finite play. It remains to show that she also wins every infinite play. Let $(C_i)_{i \in \mathbb{N}}$ be a play of the acceptance game for $\mathcal{A}_\varphi$ where $\mathcal{V}$ plays with this strategy. We have to show that the highest priority of any fixpoint that appears infinitely often on the unfolding tree $\mathbb{T}$ of this play is even.

By repeated application of Lemma 4.3.8, towards this we obtain that the sequence $(\_, (Q_{X_{i_j}}, \_), \_)_{j\in\mathbb{N}}$ of fixpoint configurations on the infinite path in $\mathbb{T}$ yields an infinite sequence $(\_, (X_{i_j})_{s_{i_j}}, \_)_{j\in\mathbb{N}}$ of associated model-checking game positions such that $s_{i_{j+1}}$ is strictly descending from s_{i_j} with respect to $X_{i_{j+1}}$. Hence, we have an infinite strictly descending chain of μ-signatures and, by Lemma 3.2.2, there is $n \in \mathbb{N}$ and a greatest-fixpoint variable X such that $X = X_{i_{j'}}$ for infinitely many $j' \geq n$ and such that $j' \geq n$ implies that $X_{i_{j'}} = X$ or $X \succ X_{i_{j'}}$. But since $\Delta(Q_X) \geq \Delta(Q_Y)$ if $X \succ Y$, we have that the highest priority of a variable that occurs infinitely often on the infinite path in $\mathbb{T}$ is that of X, which is even since X is a greatest-fixpoint variable. Hence, $\mathcal{T}, v_I \models \mathcal{A}_\varphi$. □

Lemma 4.3.10. *For all* HFL *formulas φ, we have that $\overline{\mathcal{A}_\varphi} \equiv \mathcal{A}_{\overline{\varphi}^p}$, i.e., for all pointed LTS $\mathcal{T}, v$, we have that $\mathcal{T}, v \models \overline{\mathcal{A}_\varphi}$ if and only if $\mathcal{T}, v \models \mathcal{A}_{\overline{\varphi}^p}$.*

The main issue in this proof is that complementation of HFL formulas has effects on the offset of the priorities of fixpoint states in the associated APKA. To be precise, it is not necessarily true that if the fixpoint states associated to two fixpoint variables in φ have priorities that differ by n in $\mathcal{A}$, then the states in the APKA associated to the complement of φ have priorities that differ by the same n. Consider the minimal example

$$\varphi = \nu X.\, (\mu Y.\, \mathtt{tt}) \vee \big(\mu Z_1.\, \mathtt{tt} \wedge (\nu Z_2.\, \mathtt{tt})\big).$$

and its complement formula

$$\overline{\varphi}^p = \mu\widetilde{X}.\, (\nu\widetilde{Y}.\, \mathtt{ff}) \wedge \big(\nu\widetilde{Z_1}.\, \mathtt{ff} \vee (\mu\widetilde{Z_2}.\, \mathtt{ff})\big).$$

Then, the priorities of the fixpoint states are as follows:

$\mathcal{A}_\varphi$		$\mathcal{A}_{\overline{\varphi}^p}$	
state	priority	priority	state
Q_X	1	0	$Q_{\widetilde{X}}$
Q_Y	2	1	$Q_{\widetilde{Y}}$
Q_{Z_1}	1	0	$Q_{\widetilde{Z_1}}$
Q_{Z_2}	0	1	$Q_{\widetilde{Z_2}}$

We can see that the difference between the priorities of two fixpoint states does not necessarily stay constant when commuting complementation and translation into an APKA. However, since the priority labeling we work with stems from the syntactic ordering of fixpoint variables in an HFL formula, we can show that such a change in the difference between two priorities does not alter acceptance.

Proof of Lemma 4.3.10. Since φ is in ANF, the fixpoint states of $\overline{\mathcal{A}_\varphi}$ are $\{\overline{Q_X} \mid X \in \mathcal{X}$, where $\mathcal{X}$ is the set of fixpoint variables occurring in φ. On the other hand, the fixpoint variables of $\overline{\varphi}^p$ are $\{\widetilde{X} \mid X \in \mathcal{X}\}$ and, hence the fixpoint states of $\mathcal{A}_{\overline{\varphi}^p}$ are $\{Q_{\widetilde{X}} \mid X \in \mathcal{Q}\}$. The type of $\overline{Q_X}$ and the type of $Q_{\overline{X}^p}$ is τ^X for $X \in \mathcal{X}$. The initial state of $\overline{\mathcal{A}_\varphi}$ is $\overline{Q_{X_I}}$ and the initial state of $\mathcal{A}_{\overline{\varphi}^p}$ is $Q_{\widetilde{X_I}}$.

Given $X \in \mathcal{X}$, both $\delta(\overline{Q_X})$ and $\delta(Q_{\overline{X}^p})$ are derived from ψ_X, either via $\delta(Q_X)$ or via $\overline{\psi_X}^p$. Given a subformula in ψ_X, we present the associated subformulas in $\delta(Q_X)$, $\delta(\overline{Q_X})$, $\overline{\psi_X}^p$ and $\delta(Q_{\widetilde{X}})$ via the following table:

ψ_X	$\delta(Q_X)$	$\delta(\overline{Q_X})$	$\overline{\psi_X}^p$	$\delta(Q_{\widetilde{X}})$
ψ_X	$\mathsf{to}_\delta(\psi_X)$	$\overline{\mathsf{to}_\delta(\psi_X)}$	$\overline{\psi_X}^p$	$\mathsf{to}_\delta(\widetilde{\psi_X})$
P	P	$\overline{P}$	$\neg P$	$\overline{P}$
$\neg P$	$\overline{P}$	P	P	P
$\psi_1 \vee \psi_2$	$\mathsf{to}_\delta(\psi_1) \vee \mathsf{to}_\delta(\psi_2)$	$\overline{\mathsf{to}_\delta(\psi_1)}^\delta \wedge \overline{\mathsf{to}_\delta(\psi_2)}^\delta$	$\overline{\psi_1}^p \wedge \overline{\psi_2}^p$	$\mathsf{to}_\delta(\overline{\psi_1}^p) \wedge \mathsf{to}_\delta(\overline{\psi_1}^p)$
$\psi_1 \wedge \psi_2$	$\mathsf{to}_\delta(\psi_1) \wedge \mathsf{to}_\delta(\psi_2)$	$\overline{\mathsf{to}_\delta(\psi_1)}^\delta \vee \overline{\mathsf{to}_\delta(\psi_2)}^\delta$	$\overline{\psi_1}^p \vee \overline{\psi_2}^p$	$\mathsf{to}_\delta(\overline{\psi_1}^p) \vee \mathsf{to}_\delta(\overline{\psi_1}^p)$
$\langle a\rangle\psi'$	$\langle a\rangle\mathsf{to}_\delta(\psi')$	$[a]\overline{\psi'}^\delta$	$[a]\overline{\psi'}^p$	$[a]\mathsf{to}_\delta(\overline{\psi'}^p)$
$[a]\psi'$	$[a]\mathsf{to}_\delta(\psi')$	$\langle a\rangle\overline{\psi'}^\delta$	$\langle a\rangle\overline{\psi'}^p$	$\langle a\rangle\mathsf{to}_\delta(\overline{\psi'}^p)$
$\psi_1\,\psi_2$	$\mathsf{to}_\delta(\psi_1)\,\mathsf{to}_\delta(\psi_2)$	$\overline{\mathsf{to}_\delta(\psi_1)}^\delta\,\overline{\mathsf{to}_\delta(\psi_2)}^\delta$	$\overline{\psi_1}^p\,\overline{\psi_2}^p$	$\mathsf{to}_\delta(\overline{\psi_1}^p)\,\mathsf{to}_\delta(\overline{\psi_2})$
x_i^X	f_i^X	$\overline{f_i^X}$	$\widetilde{x_i^X}$	$f_i^{\widetilde{X}}$
Y	Q_Y	$\overline{Q_Y}$	$\widetilde{Y}$	$Q_{\widetilde{Y}}$

This table[6] suggests a mapping from the fixpoint states and lambda variables of $\overline{\mathcal{A}_\varphi}$ to the fixpoint states of $\mathcal{A}_{\overline{\varphi}^p}$ by mapping $\overline{Q_X}$ to $Q_{\widetilde{X}}$ and by mapping $\overline{f_i^X}$ to $f_i^{\widetilde{X}}$. Note, however, that this mapping is not just a renaming of one automaton to the other, since the two automata can differ slightly in their priority labeling.

Consider the APKA $\mathcal{A}'$ obtained from $\mathcal{A}_{\overline{\varphi}^p}$ via renaming of $\overline{Q_X}$ to $Q_{\widetilde{X}}$ and $\widetilde{f_i^X}$ to $f_i^{\widetilde{X}}$. Note that these renamings are clearly injective, whence they induce a bijection i from the fixpoint states of $\mathcal{A}_{\overline{\varphi}^p}$ to those of $\mathcal{A}'$, and similarly for the lambda variables. By invariance of APKA under simple renaming, we have $\mathcal{A}' \equiv \overline{\mathcal{A}_\varphi}$. Moreover, $\mathcal{A}'$ is equal to $\mathcal{A}_{\overline{\varphi}^p}$ with the sole exception of its priority labeling Δ' being potentially different from the priority labeling of $\mathcal{A}_{\overline{\varphi}^p}$. Now assume that $\mathcal{P} \in \{\mathcal{V}, \mathcal{S}\}$ has a winning strategy for $\mathcal{A}_{\overline{\varphi}^p}$. We show that the same strategy is winning for $\mathcal{A}'$ and, hence, for $\overline{\mathcal{A}_\varphi}$. Suppose $\mathcal{P}$ uses their strategy from $\mathcal{A}_{\overline{\varphi}^p}$ for $\mathcal{A}'$ in the acceptance game for some LTS $\mathcal{T}, v_I$. Clearly, if the play is finite, then $\mathcal{P}$ wins, since their strategy is also winning for $\mathcal{A}_{\overline{\varphi}^p}$ which differs only in the priority labeling. Now suppose the play goes on indefinitely. By the assumption, such an infinite play is winning for $\mathcal{P}$ under the priority labeling for $\mathcal{A}_{\overline{\varphi}^p}$, i.e., the highest priority that occurs infinitely often as label for a fixpoint configuration on the infinite path in the unfolding tree generated by the play is even if $\mathcal{P} = \mathcal{V}$ and odd if $\mathcal{P} = \mathcal{S}$. We have to show that the play is also winning for $\mathcal{P}$ under Δ', the priority labeling of $\mathcal{A}'$.

Since $\mathcal{A}_{\overline{\varphi}}$ is an APKA associated to an HFL formula $\overline{\varphi}^p$ in ANF, by Lemma 4.3.8, any two consecutive fixpoints $Q_{\widetilde{X}}$ and $Q_{\widetilde{Y}}$ on the infinite path are such that either $\widetilde{X} = \widetilde{X}$ or $\widetilde{X}$ and $\widetilde{Y}$ are comparable by $\succ$ in $\overline{\varphi}$. Hence, if two distinct fixpoint states $Q_{\widetilde{X}}$ and $Q_{\widetilde{Y}}$ occur on the infinite path and $\widetilde{X}$ and $\widetilde{Y}$ are not comparable by $\succ$ in $\overline{\varphi}^p$ then there is $Q_{\widetilde{Z}}$ with $\widetilde{Z} \succ \widetilde{X}$ and $\widetilde{Z} \succ \widetilde{Y}$ that occurs between the occurrences of $Q_{\widetilde{X}}$ and $Q_{\widetilde{Y}}$. In particular, there is a fixpoint state $Q_{\widetilde{X^*}}$ that occurs infinitely often such that all other fixpoint states that occur infinitely often are of the form

[6]Note that there is a small imprecision here with regard to the last entry in the lambda variable case, i.e., $f_i^{\widetilde{X}}$: The assumption at the beginning of the section is that a lambda variable for some fixpoint variable, i.e., $\overline{X}$ in this case, has the format $x_i^{\overline{X}}$. However, since $\overline{\psi}$ is the product of complementation, the variable's actual format is $\widetilde{x_i^X}$. For the sake of clarity we rename the variable as displayed in the table. The actual name is not important in the following.

$Q_{\widetilde{Y}}$ with $\widetilde{X^*} \succ \widetilde{Y}$ in $\overline{\varphi}^p$. Since the play is winning for $\mathcal{P}$, the priority of $Q_{\widetilde{X^*}}$ is good for them.

Now assume that $Q_{\widetilde{X}}$ and $Q_{\widetilde{Y}}$ are such that X and Y are equal or are comparable by $\succ$ in $\overline{\varphi}^p$. By definition, $i^{-1}(Q_{\widetilde{X}}) = \overline{Q_X}$ and $i^{-1}(Q_{\widetilde{Y}}) = \overline{Q_Y}$. In particular, X and Y are equal in φ if and only if $\widetilde{X}$ and $\widetilde{Y}$ are equal in $\overline{\varphi}^p$ and the same holds for $\succ$ in φ and $\overline{\varphi}^p$. Hence, the play above is such that for all fixpoints $Q_{\widetilde{Y}}$ that occur infinitely often on the infinite path of the unfolding tree of the run of $\mathcal{A}'$, we have $i^{-1}(Q_{\widetilde{Y}}) = \overline{Q_Y}$ and since $X^* \succ Y$ in φ for all other Y such that $Q_{\widetilde{Y}}$ occurs infinitely often on the path, we have that $\Delta(Q_{X^*}) \geq \Delta(Q_Y)$ for all such Y, where Δ is the priority labeling of $\mathcal{A}_\varphi$. Since the priority labeling of $\overline{\mathcal{A}_\varphi}$ is obtained by adding one to the priority labeling of $\mathcal{A}_\varphi$, and since the priority labeling of $\mathcal{A}'$ is obtained by applying i^{-1} and then taking the priority labeling of $\overline{\mathcal{A}_\varphi}$, we have that the highest priority that occurs infinitely often under the priority labeling of $\mathcal{A}'$ is that of $\overline{Q_{X^*}}$. Moreover, by construction, this priority is good for $\mathcal{P}$. □

Theorem 4.3.11. *For every* HFL *formula φ in ANF and of order n there is an* APKA *$\mathcal{A}_\varphi$ such that, for every pointed LTS $\mathcal{T}, v_I$, we have that $\mathcal{T}, v_I \models \varphi$ if and only if $\mathcal{T}, v_I \models \mathcal{A}_\varphi$.*

Proof. Without loss of generality, φ is already in ANF. Let $\mathcal{T}, v_I$ be an LTS. By Lemma 4.3.9, if $\mathcal{T}, v_I \models \varphi$ then $\mathcal{T}, v_I \models \mathcal{A}_\varphi$. On the other hand, if $\mathcal{T}, v_I \not\models \varphi$, then by Lemma 3.1.4, we have that $\mathcal{T}, v_I \models \overline{\varphi}^p$. By Lemma 4.3.9, also $\mathcal{T}, v_I \models \mathcal{A}_{\overline{\varphi}^p}$ and by Lemma 4.3.10, we have that $\mathcal{A}_{\overline{\varphi}^p} \equiv \overline{\mathcal{A}_\varphi}$, so $\mathcal{T}, v_I \models \overline{\mathcal{A}_\varphi}$ and, by Lemma 4.2.22, we have that $\mathcal{T}, v_I \not\models \mathcal{A}_\varphi$. We conclude that $\mathcal{T}, v_I \models \varphi$ if and only if $\mathcal{T}, v_I \models \mathcal{A}_\varphi$. □

Remark 4.3.12. Similar to the translation from $\mathcal{L}_\mu$ to APT, the translation from φ to $\mathcal{A}_\varphi$ does not produce a notable blowup. If φ has n fixpoint variables, then $\mathcal{A}_\varphi$ has n fixpoint states. Moreover, the size of $\mathcal{A}_\varphi$ coincides with the size of φ. Finally, $\mathcal{A}_\varphi$ can be constructed in linear time in the size of φ; the only part of the construction that is not straight-forward is the priority labeling, which can be solved by a simple bottom-up greedy algorithm following the definition above.

Note, however, that the translation requires φ to be in ANF. If it is already in NNF, then, by Lemma 3.1.21, it can be brought into ANF with a polynomial blowup. Hence, $\mathcal{A}_\varphi$ is of size polynomial in the size of φ. If φ is not in NNF, then $\mathcal{A}_\varphi$ potentially is exponentially larger than φ due to the blowup incurred by NNF.

Corollary 4.3.13. *By Theorem 4.3.11, we also obtain soundness of the* HFL *model-checking game, i.e., that $\mathcal{V}$ wins the* HFL *model-checking game for a formula φ starting from $\mathcal{T}, v$ only if $\mathcal{T}, v \models \varphi$.*

This previous corollary holds due to the following: If $\mathcal{V}$ wins the HFL model-checking game for φ from $\mathcal{T}, v$, then, by the proof of Lemma 4.3.9, $\mathcal{V}$ can obtain a winning strategy for the acceptance game of $\mathcal{A}_\varphi$ from $\mathcal{T}, v$. By Theorem 4.3.11, we obtain that $\mathcal{T}, v \models \varphi$.

4.4 From APKA to HFL

The topic of this section is the opposite of that of Section 4.3, namely to show that for every APKA $\mathcal{A}$ there is an equivalent HFL formula $\varphi_{\mathcal{A}}$. There are clear parallels to the strategy used in Section 4.3 in that again we use the HFL model-checking game and strategy mappings to show that $\mathcal{V}$ can leverage a winning strategy for the model-checking game into a winning strategy for the APKA acceptance game. We will also rely on the fact that complementation commutes with the translation from APKA to HFL to reduce duplicate arguments.

However, similar to the situation with parity automata and $\mathcal{L}_\mu$ (cf. Section 2.2.5), for this direction there is an additional blowup. Precedence between fixpoint variables in an HFL formula is derived from $\succ$, which depends on the position of the variables in the syntax tree. In particular, this is a partial order, and this order restricts which variables can appear in the defining formula of another variable. On the other hand, precedence between fixpoint states in an APKA is due to the priority labeling, which generally does not induce a particular structure, or even an order, on the fixpoint states. Moreover, any fixpoint state can appear in the transition relation of any other fixpoint state. Reducing the behavior of APKA onto the behavior of HFL requires the same additional unrolling technique seen in the context of parity automata, i.e., for each fixpoint variable encoding a fixpoint state, we must trace the set of fixpoint states already encoded by fixpoint variables that are outermore in the formula we construct. Moreover, we have to account for the parity condition in this construction, which means that only some fixpoint states count towards this set. Potentially, this introduces a blowup of exponential size (cf. also [18] for the same problem on parity automata). Since we also need to produce a well-named formula in order to use the HFL model-checking game, fixpoint variables have to be annotated by said set in order to make them unique.

In order to illustrate these challenges, consider the APKA $\mathcal{A}$ defined via the following system of equations:

$$\begin{aligned}
I\colon () &\mapsto_0 X\,N\\
N\colon (y\colon \bullet) &\mapsto_0 \langle a\rangle y\\
X\colon (f\colon \bullet\to\bullet) &\mapsto_1 (f\,P \land Y) \lor X\,(D\,f)\\
D\colon (g\colon \bullet\to\bullet, z\colon\bullet) &\mapsto_0 g\,(g\,z)\\
Y\colon () &\mapsto_2 X\,N
\end{aligned}$$

It accepts any vertex in an LTS if there is an infinite path from the vertex and an infinite sequence $d_1, d_2, \ldots$ of powers of 2 such that P holds on this path after d_1, $d_1 + d_2, \ldots$ many steps. A naïve translation would yield the following formulas (with type annotations omitted):

$$\begin{aligned}
\varphi_I &= \mu I.\, \varphi_X\, \varphi_N\\
\varphi_N &= \mu N.\, \lambda y.\, \langle a\rangle y\\
\varphi_X &= \mu X.\, \lambda f.\, (f\,P \land \varphi_Y) \lor X\,(\varphi_D\, f)\\
\varphi_D &= \mu D.\, \lambda g.\, \lambda z.\, g\,(g\,z)\\
\varphi_Y &= \nu Y.\, X\,N
\end{aligned}$$

Putting it all together, we obtain the formula

$$\mu I.\, \Big(\mu X.\, \lambda f.\, (f\,P \land \nu Y.\, X\,\big(\mu N.\, \lambda y.\, \langle a\rangle y\big)) \lor X\,\big((\mu D.\, \lambda g.\, \lambda z. g\,(g\,z))\, f\big)\Big)$$
$$\mu N.\, \lambda y.\, \langle a\rangle y.$$

However, the priority between the fixpoints, respectively fixpoint variables, induced by the parity labeling is clearly not reflected in this formula, since the fixpoint X, which corresponds to a fixpoint state of priority 1, is outermore than the fixpoint Y, which corresponds to a fixpoint state of priority 2. This also holds for other fixpoints, e.g., I, but X and Y are clearly mutually recursive. The solution is to unfold X in order to restore the syntactical order of the fixpoints to that induced by the priority labeling. Hence, we obtain the following formula:

$$\mu I.\, \Big(\mu X.\, \lambda x.\, (f\, P \wedge \nu Y.\, \varphi_{X'}\, \varphi_N) \vee X\, \big((\mu D.\, \lambda g.\, \lambda z. g\, (g\, z))\, f\big)\Big)\, \varphi_N$$

where

$$\varphi_{X'} = \mu X'.\, \lambda x.\, (f\, P \wedge Y) \vee X'\, \big((\mu D.\, \lambda g.\, \lambda z. g\, (g\, z))\, f\big)$$

and $\varphi_N = \mu N.\, \lambda y.\, \langle a \rangle y$. In this formula, Y does not depend on X, but rather on X', but now Y is outermore than X', which is necessary to encode the priority labeling faithfully.

The result of all this is that both the formulas we work with become somewhat more unwieldy than it was the case in Section 4.3, and that some of the steps, in particular around the acceptance condition in the correctness proof, become more involved.

4.4.1 Preparations

For the remainder of the section, unless explicitly stated otherwise, fix the APKA $\mathcal{A} = (\mathcal{Q}, \Delta, Q_I, \delta, (\tau_Q)_{Q \in \mathcal{Q}})$ with fixpoint state set $\mathcal{Q}$. Let $\tau_Q = \tau_1^Q \to \cdots \to \tau_{k_Q}^Q \to \bullet$ for all $Q \in \mathcal{Q}$, and let the respective lambda variables of the fixpoint states be $(f_1^Q : \tau_1), \dots, (f_{k_Q}^Q : \tau_{k_Q})$ with the indicated types. Let $<_{\mathcal{Q}}$ be a strict total order on $\mathcal{Q}$ that is compatible with the Δ in the sense that, if $Q <_{\mathcal{Q}} Q'$ then $\Delta(Q) \leq \Delta(Q')$. The purpose of this ordering is to bring the fixpoint state set of $\mathcal{A}$ as close as possible to the situation where each state has its own priority labeling. The reason for this is that we unravel the transition structure of $\mathcal{A}$ into an HFL formula, where individual fixpoint variables are annotated by the set of variables that appear free in their defining formula. However, since these variables themselves are annotated, we need to know the exact annotation of such a variable that occurs freely. If the variables are totally ordered, we can pretend that their priority labeling follows this order. This then makes the annotation of a variable that appears freely in the defining formula of another unique, since the annotation is obtained as the collection of all variables that are not lower in the total order. Without such a total ordering, the annotations would depend on the exact path in which the original variable was reached. Annotation by such a path is possible, but unwieldy.

Lemma 4.4.1. *Consider the pair $(O, <_{\mathcal{A}})$ where $O = \{\mathcal{R} \mid \mathcal{R} \subseteq \mathcal{Q}\}$ and $\mathcal{R}_1 <_{\mathcal{A}} \mathcal{R}_2$ if there is $Q \in \mathcal{Q}$ such that*

1. *$Q \notin \mathcal{R}_1$ but $Q \in \mathcal{R}_2$, and*
2. *For all $Q' \in \mathcal{Q}$ with $Q' >_{\mathcal{Q}} Q$ we have that if $Q' \in \mathcal{R}_1$, then $Q' \in \mathcal{R}_2$.*

Then $(O, <_{\mathcal{A}})$ is a strict partial order, i.e., it is irreflexive, antisymmetric and transitive. Moreover, $\emptyset$ and $\mathcal{Q}$ are the least and greatest elements of this order. Finally, the longest ascending chain in this order has length at most $|2^{\mathcal{Q}}|$.

Proof. Irreflexivity follows from the first item. For transitivity, let $\mathcal{R}_2 <_{\mathcal{A}} \mathcal{R}_1$ and $\mathcal{R}_3 <_{\mathcal{A}} \mathcal{R}_2$. Then there is Q_1 such that

- $Q_1 \in \mathcal{R}_1$ but $Q_1 \notin \mathcal{R}_2$, and
- for all $Q' >_{\mathcal{Q}} Q_1$ we have that if $Q' \in \mathcal{R}_1$ then $Q' \in \mathcal{R}_2$.

Moreover, there is Q_2 such that

- $Q_2 \in \mathcal{R}_2$ but $Q_1 \notin \mathcal{R}_3$, and
- for all $Q' >_{\mathcal{Q}} Q_2$ we have that if $Q' \in \mathcal{R}_2$ then $Q' \in \mathcal{R}_3$.

Let Q be the maximum of Q_1 and Q_2. Then, for all $Q' >_{\mathcal{Q}} Q$ we have that if $Q' \in \mathcal{R}_1$ then $Q' \in \mathcal{R}_2$ and, hence, also $Q' \in \mathcal{R}_3$. This settles the second condition to show that $\mathcal{R}_3 <_{\mathcal{A}} \mathcal{R}_1$. Towards the first condition, there are two cases: If $Q = Q_1$ then $Q \in \mathcal{Q}_1$ but $Q \notin \mathcal{R}_2$. Since $Q >_{\mathcal{A}} Q_2$, also $Q \notin \mathcal{R}_3$, for by contraposition of the second item. If $Q = Q_2$, then, by the second item, $Q \in \mathcal{R}_1$.

Antisysmmetry follows from irreflexivity and transitivity. The claim on $\emptyset$ and $\mathcal{Q}$ being the least and greatest elements of $<_{\mathcal{A}}$ is a straightforward verification.

The claim on the length of the longest chain in this order follows from the fact that the underlying set of the order is $2^{\mathcal{Q}}$. □

Given a subset $\mathcal{R} \subseteq \mathcal{Q}$ and some $Q \in \mathcal{Q}$, define $\mathcal{R} \upharpoonright Q$ as $\big(\mathcal{R} \setminus \{Q' \mid Q' <_{\mathcal{Q}} Q\}\big) \cup \{Q\}$. Note that, in particular, for all $Q' \in \mathcal{R} \upharpoonright Q$, we have that $\Delta(Q') \geq \Delta(Q)$ due to the definition of $<_{\mathcal{Q}}$. Hence, if $Q \notin \mathcal{R}$, we have that $\mathcal{R} <_{\mathcal{A}} \mathcal{R} \upharpoonright Q$.

The upper bound on the length on ascending chains in $(\mathcal{Q}, <_{\mathcal{A}})$ is actually tight. The chain is obtained by starting from the empty set and, in each step, adding the least state with respect to $<_{\mathcal{Q}}$ that is not already in the previous set while simultaneously restricting to that state. Think of incrementing binary numbers, where the bits are the fixpoint states in the sets of the chain. Here, Q_1 is the least significant bit while Q_n is the most significant bit. For example, if $n = 3$ and $Q_1 <_{\mathcal{Q}} Q_2 <_{\mathcal{Q}} Q_3$ then the chain is

$$\begin{aligned}\emptyset <_{\mathcal{A}} \{Q_1\} <_{\mathcal{A}} \{Q_2\} <_{\mathcal{A}} \{Q_1, Q_2\} <_{\mathcal{A}} \{Q_3\} \\ <_{\mathcal{A}} \{Q_1, Q_3\} <_{\mathcal{A}} \{Q_2, Q_3\} <_{\mathcal{A}} \{Q_1, Q_2, Q_3\} = \mathcal{Q}.\end{aligned}$$

4.4.2 Definition of $\varphi_{\mathcal{A}}$

Let

$$\mathcal{X}_{\mathcal{A}} = \{X_{Q,\mathcal{R}} \mid Q \in \mathcal{Q}, \mathcal{R} \subseteq \mathcal{Q}, Q \in \mathcal{R}\}$$

be a set of HFL fixpoint variables and let

$$\mathcal{F}_{\mathcal{A}} = \bigcup_{X_{Q,\mathcal{R}} \in \mathcal{X}_{\mathcal{A}}} \{x_1^{Q,\mathcal{R}}, \ldots, x_{k_Q}^{Q,\mathcal{R}}\}$$

be a set of HFL lambda variables.

Associate with each fixpoint state $X_{Q,\mathcal{R}} \in \mathcal{X}_{\mathcal{A}}$ an HFL formula $\varphi_{Q,\mathcal{R}}$ defined inductively over its transition relation via

$$\varphi_{Q,\mathcal{R}} = \sigma_Q(X_{Q,\mathcal{R}} \colon \tau_Q).\, \lambda(x_1^{Q,\mathcal{R}} \colon \tau_1^Q).\, \ldots \lambda(x_{k_Q}^{Q,\mathcal{R}} \colon \tau_{Q_i}^Q).\, \mathsf{toHFL}_{Q,\mathcal{R}}(\delta(Q))$$

where $\sigma_Q = \mu$ if $\Delta(Q)$ is odd, and $\sigma_Q = \nu$ if $\Delta(Q)$ is even, and where $\mathsf{toHFL}_{Q,\mathcal{R}}(\delta(Q))$ is

$$\delta(Q)[\neg P/\overline{P} \mid P \in \mathcal{P}][x_j^{Q,\mathcal{R}}/f_j^Q \mid 1 \leq j \leq k_Q]$$
$$[\varphi_{Q',\mathcal{R}\restriction Q'}/Q' \mid Q' \notin \mathcal{R}][X_{Q',\mathcal{R}\restriction Q'}/Q' \mid Q' \in \mathcal{R}].$$

In other words, $\mathsf{toHFL}_{Q,\mathcal{R}}(\delta(Q))$ is defined inductively via

$$\begin{aligned}
\mathsf{toHFL}_{Q,\mathcal{R}}(P) &= P\\
\mathsf{toHFL}_{Q,\mathcal{R}}(\overline{P}) &= \neg P\\
\mathsf{toHFL}_{Q,\mathcal{R}}(\chi \vee \zeta) &= \mathsf{toHFL}_{Q,\mathcal{R}}(\chi) \vee \mathsf{toHFL}_{Q,\mathcal{R}}(\zeta)\\
\mathsf{toHFL}_{Q,\mathcal{R}}(\chi \wedge \zeta) &= \mathsf{toHFL}_{Q,\mathcal{R}}(\chi) \wedge \mathsf{toHFL}_{Q,\mathcal{R}}(\zeta)\\
\mathsf{toHFL}_{Q,\mathcal{R}}(\langle a\rangle \chi) &= \langle a\rangle \mathsf{toHFL}_{Q,\mathcal{R}}(\chi)\\
\mathsf{toHFL}_{Q,\mathcal{R}}([a] \chi) &= [a] \mathsf{toHFL}_{Q,\mathcal{R}}(\chi)\\
\mathsf{toHFL}_{Q,\mathcal{R}}(\chi\, \zeta) &= \mathsf{toHFL}_{Q,\mathcal{R}}(\chi)\, \mathsf{toHFL}_{Q,\mathcal{R}}(\zeta)\\
\mathsf{toHFL}_{Q,\mathcal{R}}(f_j^Q) &= x_j^{Q,\mathcal{R}} \text{ for } 1 \leq j \leq k_Q\\
\mathsf{toHFL}_{Q,\mathcal{R}}(Q') &= \varphi_{Q',\mathcal{R}\restriction Q'} \text{ if } Q' \notin \mathcal{R}\\
\mathsf{toHFL}_{Q,\mathcal{R}}(Q') &= X_{Q',\mathcal{R}\restriction Q'} \text{ if } Q' \in \mathcal{R}.
\end{aligned}$$

Lemma 4.4.2. *For all $\emptyset \neq \mathcal{R}, \mathcal{R}_1, \mathcal{R}_2 \in \mathcal{Q}$ and for all $Q \in \mathcal{R}$, the following are true:*

1. *If $\varphi_{Q_1,\mathcal{R}_1}$ contains $\varphi_{Q_1,\mathcal{R}_2}$ as proper subformula, then $\mathcal{R}_1 <_{\mathcal{A}} \mathcal{R}_2$.*
2. *$\varphi_{Q,\mathcal{R}}$ has no free lambda variables, and its free fixpoint variables are from the set*
$$\{X_{Q',\mathcal{R}'} \mid Q' \in \mathcal{R} \setminus \{Q\}, \mathcal{R}' = \mathcal{R} \restriction Q'\}.$$
3. *Let $\Sigma^{\mathcal{R}}$ contain the set of typing hypotheses $\{X_{Q',\mathcal{R}\restriction Q'} \colon \tau_{Q'} \mid Q' \in \mathcal{R} \setminus \{Q\}\}$. Then the typing judgment $\Sigma^{\mathcal{R}} \vdash \varphi_{Q,\mathcal{R}} \colon \tau_Q$ is derivable.*

Moreover, $\varphi_{Q_I,\{Q_I\}}$ is well-defined and of size $\mathcal{O}(n \cdot 2^n \cdot d)$ where $n = |\mathcal{Q}|$ and d is the size $\mathcal{A}$.

Proof. We prove the itemized claims first. The proof will be by induction over $<_{\mathcal{A}}$, starting from its maximal element. For the induction start, consider $\varphi_{Q,\mathcal{Q}}$ where Q is arbitrary. Since $\mathcal{Q}$ contains all fixpoint states, all occurrences of a fixpoint state Q' in the transition relation of Q will be substituted by $X_{Q',\mathcal{Q}\restriction Q'}$. Hence, $\varphi_{Q,\mathcal{Q}}$ contains no formula of the form $\varphi_{Q',\mathcal{R}'}$ as a proper subformula. It also follows that the free fixpoint variables of $\varphi_{Q,\mathcal{Q}}$ are as in the lemma, and the claim on lambda variables follows from the definition of $\varphi_{Q,\mathcal{Q}}$.

Now let Σ' contain the typing hypotheses $Q' \colon \tau_{Q'}$ for all $Q' \in \mathcal{Q}$, as well as the hypotheses $f_i^Q \colon \tau_i^Q$ for all $1 \leq i \leq k_Q$. Then by well-typedness of $\mathcal{A}$, the judgment $\Sigma' \vdash \delta(Q) \colon \bullet$ is derivable. Let $\Sigma_\lambda^{\mathcal{R}}$ contain all typing hypotheses that $\Sigma^{\mathcal{R}}$ does, as well as the hypotheses $x_1^{Q,\mathcal{R}} \colon \tau_1^Q, \ldots, x_{k_Q}^{Q,\mathcal{R}} \colon \tau_{k_Q}^Q$. Then the judgment $\Sigma' \vdash \delta(Q) \colon \bullet$ can be used to obtain the judgment $\Sigma_\lambda^{\mathcal{R}} \vdash \mathsf{toHFL}_{Q,\mathcal{R}}(\delta(Q)) \colon \bullet$ by replacing the APKA variables by their HFL counterparts and accounting for the switch from $\overline{P}$ to $\neg P$.

The rest of the typing claim follows by k_Q-fold application of the typing rule for lambda abstraction followed by an application of the typing rule for fixpoints.

Now consider $\varphi_{Q,\mathcal{R}}$ and assume that we have proved the lemma for all $\varphi_{Q',\mathcal{R}'}$ with $\mathcal{R} <_{\mathcal{A}} \mathcal{R}'$. The itemized claims now all follow relatively easily: Regarding the subformula claim, note that $\varphi_{Q,\mathcal{R}}$ contains $\varphi_{Q',\mathcal{R}'}$ as a substitution instance of Q' in $\delta(Q)$ only if $Q' \notin \mathcal{R}$. Hence, $\mathcal{R} <_{\mathcal{A}} \mathcal{R} \upharpoonright Q' = \mathcal{R}'$, since $\mathcal{R}'$ contains Q' and $\mathcal{R}$ does not. By applying the induction hypothesis to $\varphi_{Q',\mathcal{R}'}$ and invoking transitivity of $<_{\mathcal{A}}$, we obtain Item 1 for $\varphi_{Q,\mathcal{R}}$. Moreover, since, by the induction hypothesis, $\varphi_{Q',\mathcal{R}'}$ has no free lambda variables, and its free fixpoint variables are in $\mathcal{R}' \setminus \{Q'\}$ which is a subset of $\mathcal{R}$. Hence, Item 2 follows as well. Moreover, it also follows that all typing hypotheses in $\Sigma^{\mathcal{R}'}$ are also in $\Sigma^{\mathcal{R}}$, whence the judgment $\Sigma^{\mathcal{R}} \vdash \varphi_{Q',\mathcal{R}'} \colon \tau_Q$ is derivable. By constructing a derivation for $\varphi_{Q,\mathcal{R}}$ from that for $\delta(Q)$ and by using the type derivations for $\varphi_{Q',\mathcal{R}'}$ that occur as proper subformulas in $\varphi_{Q,\mathcal{R}}$, we can also construct a derivation for $\Sigma^{\mathcal{R}} \vdash \varphi_{Q,\mathcal{R}} \colon \tau_Q$.

It follows that $\varphi_{Q_I,\{Q_I\}}$ is well-defined. Regarding the size estimate, note that there are at most $2^n \cdot n$ many formulas of the form $\varphi_{Q,\mathcal{R}}$ if n is the size of $\mathcal{Q}$. Moreover, the size of each $\varphi_{Q,\mathcal{R}}$ is bounded by d if we consider proper subformulas of the form $\varphi_{Q',\mathcal{R}'}$ of unit size. Hence, the result on the size of $\varphi_{Q_I,\{Q_I\}}$ follows if each of these subformulas occurs at most once in a DAG representation of $\varphi_{Q_I,\{Q_I\}}$. For the sake of contradiction, assume that this is not the case. Then there are two occurrences of some subformula $\varphi_{Q,\mathcal{R}}$ that cannot be identified in such a DAG representation. The reason for this is that there is a free variable of $\varphi_{Q,\mathcal{R}}$ that is bound in two nodes of the syntax tree that cannot be identified themselves. But these two binders are binders of $\varphi_{Q',\mathcal{R}'}$ such that $\mathcal{R}' = \mathcal{R} \upharpoonright Q$ by Item 2. In particular, $\mathcal{R}' <_{\mathcal{A}} \mathcal{R}$. Since $<_{\mathcal{A}}$ is finite and, hence, well-founded, repeated application of this argument yields a contradiction. It follows that each subformula of the form $\varphi_{Q,\mathcal{R}}$ occurs at most once in $\varphi_{Q_I,\{Q_I\}}$ which, hence, can be seen to have size in $\mathcal{O}(2^n \cdot n \cdot d)$. ☐

Definition 4.4.3. Given an APKA $\mathcal{A}$, we define the HFL formula $\varphi_{\mathcal{A}}$ as $\varphi_{Q_I}^{\{Q_I\}}$ as per above.

Note that, if $\mathcal{A}$ has order k then so does $\varphi_{\mathcal{A}}$, by Item 3 of Lemma 4.4.2.

Before we proceed to the correctness proof, i.e., the proof that exactly those pointed LTS are models of $\varphi_{\mathcal{A}}$ that are accepted by $\mathcal{A}$, we show that, similar to the situation for the inverse direction, the translation commutes with complementation.

Lemma 4.4.4. *For all* APKA $\mathcal{A}$*, we have that* $\varphi_{\overline{\mathcal{A}}} \equiv \overline{\varphi_{\mathcal{A}}}^p$.

Proof. We proceed similarly to the proof of Lemma 4.3.10 by showing that $\varphi_{\overline{\mathcal{A}}}$ is just a renaming of $\overline{\varphi_{\mathcal{A}}}^p$. Note that $\overline{\varphi_{\mathcal{A}}}^p$ is in ANF by Observation 3.1.22. For a set $\mathcal{R}$ of fixpoint states of $\mathcal{A}$, we write $\overline{\mathcal{R}}$ for the set $\{\overline{Q} \mid Q \in \mathcal{R}\}$. Note that, by the definition of $\overline{\mathcal{A}}$, we have that $\overline{\Delta}(\overline{Q}) = \overline{\Delta}(\overline{Q'}) + k$ if and only if $\Delta(Q) = \Delta(Q') + k$ Moreover, $\overline{Q'}$ occurs in $\overline{\delta}(\overline{Q})$ if and only if Q' occurs in $\delta(Q)$ since $\overline{\delta}(\overline{Q}) = \overline{\delta(Q)}^{\delta}$. Hence, $\overline{X_{Q,\mathcal{R}}}$ appears as a fixpoint variable in $\overline{\varphi_{\mathcal{A}}}^p$ if and only if $X^{\overline{Q},\overline{\mathcal{R}}}$ is a fixpoint variable in $\varphi_{\overline{\mathcal{A}}}$, and the defining formula of $X_{\overline{Q},\overline{\mathcal{R}}}$ is obtained from that of $\widetilde{X_{Q,\mathcal{R}}}$ via the renaming of the fixpoint variables and lambda variables of $\overline{\varphi_{\mathcal{A}}}^p$ into the corresponding variables of $\varphi_{\overline{\mathcal{A}}}$ that renames $\widetilde{X_{Q,\mathcal{R}}}$ into $X_{\overline{Q},\overline{\mathcal{R}}}$ and $\widetilde{x_j^{Q,\mathcal{R}}}$ into $x_j^{\overline{Q},\overline{\mathcal{R}}}$. Since HFL is invariant under renaming of variables, we obtain that $\overline{\varphi_{\mathcal{A}}} \equiv \varphi_{\overline{\mathcal{A}}}$. ☐

4.4.3 The Correctness Proof

In order to prove the equivalence of $\varphi_{\mathcal{A}}$ and $\mathcal{A}$, we use the HFL model-checking game defined in Section 3.2 again. Given an LTS $\mathcal{T}$ such that $\mathcal{T}, v_I \models \varphi_{\mathcal{A}}$, player $\mathcal{V}$ will use her winning strategy in the model-checking game for $\mathcal{T}, v_I$ and $\varphi_{\mathcal{A}}$ to construct an accepting run of $\mathcal{A}$ starting from $\mathcal{T}, v_I$. This is done by keeping the current position in the model-checking game similar via a strategy mapping to the current configuration in the acceptance game. If $\mathcal{V}$ has to make a decision in the acceptance game, she consults her strategy in the model-checking game in order to obtain a good choice. If $\mathcal{S}$ is to make a decision in the acceptance game, $\mathcal{V}$ mimics this move in the model-checking game in order to keep the positions synchronized.

Recall Definition 4.3.5: Let $(v, (\chi, e), \Gamma)$ and $(v', \psi, \digamma)$ be a configuration in the acceptance game for some APKA, respectively the model-checking game for some HFL formula. Let $c_1, \dots, c_k$ and $\psi_1, \dots, \psi_{k'}$ be the contents of Γ and $\digamma$ from top to bottom. We say that the pair is *good* if:

1. $\mathcal{V}$ wins from $(v', \psi, \digamma)$, and
2. $v = v'$, and
3. $k = k'$ and,
4. there are *strategy mappings* $m, m_1, \dots, m_k$ from the syntax trees of ψ and $\psi_1, \dots, \psi_k$ to (χ, e), respectively to $c_1, \dots, c_k$ that agree on fixpoint variables.

Given two configurations C_1 and C_2 of the APKA acceptance game for some APKA, and two positions P_1 and P_2 in the HFL model-checking game for some HFL formula such that C_2 is a valid successor of C_1 and P_2 is a valid successor of P_1, we say that the pair of transitions if *good* if the pair of C_i and P_i is good for $i \in \{1, 2\}$ and if all strategy mappings involved agree on fixpoints.

Lemma 4.4.5. *Let $P = (v, X^s_{Q,\mathcal{R}}, \digamma)$ be a position in the* HFL *model-checking game for $\varphi_{\mathcal{A}}$ such that $\mathcal{V}$ has a winning strategy from P. Let $\psi_1, \dots, \psi_k$ be the contents of $\digamma$ from top to bottom.*

If $X_{Q,\mathcal{R}}$ is a greatest-fixpoint variable, let s' be defined as

$$\begin{aligned} s'(X_{Q',\mathcal{R}'}) &= s(X_{Q',\mathcal{R}'}) && \text{if } Q' >_{\mathcal{Q}} Q, \mathcal{R}' = \mathcal{R} \upharpoonright Q' \\ s'(X_{Q',\mathcal{R}}) &= \mathit{ht}(\llbracket \tau_Q \rrbracket_{\mathcal{T}}) && \text{otherwise.} \end{aligned}$$

If $X_{Q,\mathcal{R}}$ is a least-fixpoint variable, then there is an ordinal α, and a μ-signature s' defined as

$$\begin{aligned} s'(X_{Q',\mathcal{R}'}) &= \alpha && \text{if } X_{Q',\mathcal{R}'} = X_{Q,\mathcal{R}} \\ s'(X_{Q',\mathcal{R}'}) &= s(X_{Q',\mathcal{R}'}) && \text{if } Q' >_{\mathcal{Q}} Q, \mathcal{R}' = \mathcal{R} \upharpoonright Q' \\ s'(X_{Q',\mathcal{R}}) &= \mathit{ht}(\llbracket \tau_Q \rrbracket_{\mathcal{T}}) && \text{otherwise.} \end{aligned}$$

In either case, $P' = (v, \psi', \varepsilon)$ is a valid successor configuration of P that is winning for $\mathcal{V}$, where

$$\psi' = \delta(Q)[\neg P/\overline{P} \mid P \in \mathcal{P}][\psi_i/f_i^{Q,\mathcal{R}} \mid 1 \le i \le k_Q][X^{s'}_{Q',\mathcal{R}\upharpoonright Q'}/Q' \mid Q' \in \mathcal{Q}]$$

Moreover, s' is descending from s with respect to $X_{Q,\mathcal{R}}$.

Proof. If $X_{Q,\mathcal{R}}$ is a greatest-fixpoint variable, let s' be defined as in the lemma. If $X_{Q,\mathcal{R}}$ is a least-fixpoint variable, note that $s(X_{Q,\mathcal{R}}) \neq 0$ since otherwise $\mathcal{V}$ loses the model-checking game contradicting the assumption that she wins from P. If $s(X_{Q,\mathcal{R}})$ is a limit ordinal, let α be the ordinal that $\mathcal{V}$ picks in accordance with her winning strategy, if $s(X_{Q,\mathcal{R}})$ is a successor ordinal, let α be determined by $s(X_{Q,\mathcal{R}}) = \alpha + 1$. Let s' be defined as in the lemma. The statement that s' is descending from s with respect to $X_{Q,\mathcal{R}}$ is now a straightforward verification of the definition.

For the rest of the statement, recall that

$$\begin{aligned}\mathsf{toHFL}_{Q,\mathcal{R}}(\delta(Q)) = &\\ &\delta(Q)[\neg P/\overline{P} \mid P \in \mathcal{P}][x_j^{Q,\mathcal{R}}/f_j^Q \mid 1 \leq j \leq k_Q]\\ &\qquad [X_{Q',\mathcal{R}\restriction Q'}/Q' \mid Q' \in \mathcal{R}][\varphi_{Q',\mathcal{R}\restriction Q'}/Q' \mid Q' \notin \mathcal{R}].\end{aligned}$$

and that the defining formula of the variable $X_{Q,\mathcal{R}}$ is

$$\psi_{Q,\mathcal{R}} = \sigma_Q X_{Q,\mathcal{R}}.\, \lambda x_1^{Q,\mathcal{R}}, \ldots, \lambda x_{k_Q}^{Q,\mathcal{R}}.\, \mathsf{toHFL}_{Q,\mathcal{R}}(\delta(Q)).$$

Consider the definition of the HFL model-checking game. Recall that both occurrences of fixpoint variables of the form $X_{Q',\mathcal{R}'}$ and of defining formulas of the form $\varphi_{Q',\mathcal{R}'}[Z^{s'}/Z \mid Z \in \mathcal{X}_1]$ (where $\mathcal{X}_1$ is a subset of the fixpoint variables that occur in $\varphi_{Q',\mathcal{R}'}$, and Z also has the form $X_{Q'',\mathcal{R}''}$) in $\psi_{Q,\mathcal{R}}$ are replaced by fixpoint variables annotated by s'. Either of these gets replaced to $X^{s'}_{Q,\mathcal{R}}$ by the definition of the HFL model-checking game. Hence, we obtain that ψ' is

$$\begin{aligned}\psi_{Q,\mathcal{R}}[\neg P/\overline{P} \mid P \in \mathcal{P}][\psi_i/x_i^{Q,\mathcal{R}} \mid 1 \leq i \leq k_Q][X^{s'}_{Q',\mathcal{R}\restriction Q'}/X_{Q',\mathcal{R}\restriction Q'} \mid Q' \in \mathcal{R}]&\\ [X^{s'}_{Q',\mathcal{R}\restriction Q'}/\varphi_{Q',\mathcal{R}\restriction Q'} \mid Q' \notin \mathcal{R}]&\end{aligned}$$

which then simplifies to the claim of the lemma. This position is winning for $\mathcal{V}$ since either the game is deterministic anyway, or $\mathcal{V}$ picks α according to her winning strategy. □

Lemma 4.4.6. *Let $\mathcal{T}, v_I$ be an LTS such that $\mathcal{T}, v_I \models \varphi_{\mathcal{A}}$. The pair $(v_I, (Q_I, e^0), \varepsilon)$ and $(v_I, X^{s_I}_{Q_I,\{Q_I\}}, \varepsilon)$ of starting positions in the acceptance game for $\mathcal{A}$ and the* HFL *model-checking game for $\varphi_{\mathcal{A}}$ is good. Moreover, for any given good pair C, P of positions in those two games, either $\mathcal{V}$ wins both games immediately, or there are successor positions C' of C, respectively P' of P such that the pair C', P' is also good and, moreover, the transition is good.*

Proof. Since $\mathcal{T}, v_I \models \varphi_{\mathcal{A}}$ we have that $\mathcal{V}$ wins the HFL model-checking game for $\varphi_{\mathcal{A}}$. Moreover, both stacks are empty, so the conditions pertaining to them are satisfied trivially. The mapping m with $m(X^{s_I}_{Q_I,\{Q_I\}}) = (Q_I, e^0)$ clearly is a valid strategy mapping, and since the syntax tree of $X^{s_I}_{Q_I,\{Q_I\}}$ has only one node, it is automatically good. Hence, the pair of starting positions is good.

Now suppose $C = (\_, (\chi, e), \Gamma)$ and $P = (\_, \psi, \digamma)$ is a good pair of positions. Let $(c_1, \ldots, c_k)$ be the contents of Γ from top to bottom, and let $\psi_1, \ldots, \psi_k$ be the contents of $\digamma$ from top to bottom. Let m and $m_1, \ldots, m_k$ be the strategy mappings for ψ respectively $\psi_1, \ldots \psi_k$. Depending on the form of χ, we show that $\mathcal{V}$ either wins both games immediately, or that there is a good pair of transitions available:

- If $\chi = P$ or $\chi = \neg P$, then also $\psi = P$, respectively $\psi = \overline{P}$ since $m(\psi) = \mathsf{expand}(\chi, e) = (\chi, e)$. Since $\mathcal{V}$ wins in the HFL model-checking game, we have that $\mathcal{T}, v \models \psi$ and, hence, also $\mathcal{T}, v \models \chi$, whence she also wins the acceptance game for $\mathcal{A}$.

- If $\chi = \chi_1 \vee \chi_2$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \psi_1 \vee \psi_2$. Since $\mathcal{V}$ has a winning strategy in the HFL model-checking game for $\varphi_{\mathcal{A}}$, she picks $i \in \{1, 2\}$ and the acceptance game for $\mathcal{A}$ continues from $C' = (v, (\chi_i, e), \Gamma)$ and the HFL model-checking game for $\varphi_{\mathcal{A}}$ continues from $P' = (v, \psi_i, \digamma)$. By assumption, $\mathcal{V}$ also wins from P'. Since both stacks have not been altered, the conditions pertaining to them and their associated strategy mappings are satisfied by assumption, respectively by keeping the strategy mappings from the pair C, P. The strategy mapping m' for ψ_i is obtained as the restriction of m to the subtree induced by the φ_i. Clearly, this is a valid and good strategy mapping. Since it agrees with m on fixpoints, the transition is good.

- If $\chi = \chi_1 \wedge \chi_2$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \psi_1 \wedge \psi_2$. Then $\mathcal{S}$ picks $i \in \{1, 2\}$ and the acceptance game for $\mathcal{A}$ continues from $C' = (v, (\chi_i, e), \Gamma)$ and the HFL model-checking game for $\varphi_{\mathcal{A}}$ continues from $P' = (v, \psi_i, \digamma)$. Since $\mathcal{V}$ has a winning strategy in the HFL model-checking game for $\varphi_{\mathcal{A}}$, $\mathcal{V}$ also wins from P'. The rest of the argument proceeds as in the previous case.

- If $\chi = \langle a \rangle \chi'$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \langle a \rangle \psi'$. Since $\mathcal{V}$ has a winning strategy in the HFL model-checking game for $\varphi_{\mathcal{A}}$, she picks w such that $v \xrightarrow{a} w$ and $(w, \psi', \digamma)$ is winning for $\mathcal{V}$. Since $\mathcal{V}$ wins from P, such w must exist. The acceptance game for $\mathcal{A}$ continues from $C' = (w, (\chi', e), \Gamma)$ and the HFL model-checking game for $\varphi_{\mathcal{A}}$ continues from $P' = (w, \psi', \digamma)$. Since both stacks have not been altered, the conditions pertaining to them and their associated strategy mappings are satisfied by assumption, respectively by keeping the strategy mappings from the pair C, P. The strategy mapping m' for ψ' is obtained as the restriction of m to the subtree induced by the φ'. Clearly, this is a valid strategy mapping, and the pair C', P' is good since the pair C, P was. Since $m'(t) = m(t1)$ for any node t in the syntax tree of ψ, the pair of transitions is also good.

- If $\chi = [a]\chi'$, from $m(\psi) = (\chi, e)$ it follows that $\psi = [a]\psi'$. Hence, $\mathcal{S}$ picks w such that $v \xrightarrow{a} w$, if possible. If not, $\mathcal{V}$ wins both games immediately. If $\mathcal{S}$ is not stuck, by assumption, $\mathcal{V}$ wins from $(w, \psi', \digamma)$. The rest of the argument proceeds as in the previous case.

- If $\chi = \chi'_1\, \chi'_2$, from $m(\psi) = (\chi, e)$ it follows that $\psi = \psi'_1\, \psi'_2$. Then the acceptance game for $\mathcal{A}$ continues in $C' = (v, (\chi'_1, e), \Gamma')$ and the HFL model-checking game for $\varphi_{\mathcal{A}}$ continues from $(v, \psi'_1, \digamma')$ where the contents of Γ are $(\chi'_2, e), c_1, \ldots, c_k$ from top to bottom, and the contents of $\digamma'$ are $\psi'_2, \psi_1, \ldots, \psi_k$ from top to bottom. Since $\mathcal{V}$ wins from P, and the game is determined, she also wins from P'. Since the stacks gained one element each, the condition pertaining to their sizes is satisfied in the pair C', P'. For the bottom k elements, the strategy mappings continue to be $m_1, \ldots, m_k$ from top to bottom. The strategy mapping m'_2 from the new topmost stack element ψ'_2 of $\digamma$ to the new topmost element (χ'_2, e) is obtained as the restriction of m to the subtree induced by ψ_2' in ψ, which is a valid strategy mapping since m is. The strategy mapping m_1' from ψ_1' to (χ_1', e) is obtained as the restriction of m to the subtree induced by ψ_1' in ψ, which is a valid strategy mapping for the same reason. Since the new strategy mappings are restrictions of strategy mappings from the pair C, P, not only is the pair C', P' good, but the transition is also good.

- If $\chi = f_i^Q$, then the acceptance game for $\mathcal{A}$ continues in the configuration $C' = (v, \mathsf{lookup}(f_i^Q, e), \Gamma)$ and the HFL model-checking game for $\varphi_{\mathcal{A}}$ remains in P. By definition, $m(\psi) = \mathsf{expand}((\chi, e)) = \mathsf{expand}(\mathsf{lookup}(f_i^Q, e))$, whence m is also a valid strategy map for ψ in the pair C', P. Since everything else stays put, the new pair is good and so is the transition.

- If $\chi = Q$, since $m(\psi) = (\chi, e)$, we have that $\psi = X^s_{Q,\mathcal{R}}$ for some $\emptyset \neq \mathcal{R} \subseteq \mathcal{Q}$, $Q \in \mathcal{R}$, and a μ-signature s. By Lemma 4.4.5, there is a successor configuration $P' = (v, \psi', \varepsilon)$ of P that is winning for $\mathcal{V}$ such that

$$\psi' = \delta(Q)\underbrace{[\neg P/\overline{P} \mid P \in \mathcal{P}][\psi_i/f_i^{Q,\mathcal{R}} \mid 1 \leq i \leq k_Q][X^{s'}_{Q',\mathcal{R}\upharpoonright Q'}/Q' \mid Q' \in \mathcal{Q}]}_{\kappa}$$

and s' is descending from s in $X_{Q,\mathcal{R}}$. In order not to repeat the long string of substitutions every time,we write $\psi''[\kappa]$ for

$$\psi''[\neg P/\overline{P} \mid P \in \mathcal{P}][\psi_i/f_i^{Q,\mathcal{R}} \mid 1 \leq i \leq k_Q][X^{s'}_{Q',\mathcal{R}\upharpoonright Q'}/Q' \mid Q' \in \mathcal{Q}]$$

if ψ'' is a subformula of ψ'.

The acceptance game for $\mathcal{A}$ continues in $C' = (v, (\delta(Q), e'), \varepsilon)$ where $e' = (f_1^Q \mapsto c_1, \ldots, f_k^Q \mapsto c_k, \_, \_)$.

We now define the new strategy mapping m' by induction over the syntax tree of ψ'. The initial mapping of m' is $m'(\psi') = (\mathsf{expand}(\delta(Q), e')$. Let t be a node in the syntax tree of the form $\psi''[\kappa]$ where ψ'' is a subformula of $\delta(Q)$. Depending on the top operator of the formula ψ'' for t, we continue to define m'. In order to improve readability, we will not explicitly display all substitutions:

  - If ψ'' is P or $\overline{P}$, there is nothing to define.
  - If ψ'' is $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$, then $m'(\psi''[\kappa]) = \mathsf{expand}(\psi'', e') = (\psi'', e')$ which is $(\psi_1 \vee \psi_2, e')$, respectively $(\psi_1 \wedge \psi_2, e')$. Define $m'(\psi_i''[\kappa]) = \mathsf{expand}(\psi'', e')$ for $i \in \{1, 2\}$.
  - If ψ'' is $\langle a \rangle \psi'''$ or $[a]\psi'''$ then we have that $m'(\psi''[\kappa]) = \mathsf{expand}(\psi'', e)$ which is $\mathsf{expand}(\langle a \rangle \psi, e') = (\langle a \rangle \psi, e')$, respectively $\mathsf{expand}([a]\psi, e') = ([a]\psi, e')$. Define $m'(\psi'''[\kappa]) = \mathsf{expand}((\psi''', e'))$.
  - If ψ'' is $\psi_1\,\psi_2$ then $m'(\psi''[\kappa]) = \mathsf{expand}((\psi'', e')) = (\psi'', e')$ which is $(\psi_1\,\psi_2, e')$. Define $m'(\psi_i''[\kappa]) = \mathsf{expand}((\psi'', e'))$ for $i \in \{1, 2\}$.
  - If ψ'' is Q' then $\psi''[\kappa] = X^s_{Q',\mathcal{R}\upharpoonright Q'}$ and $m'(\psi''[\kappa]) = (Q', e')$. There is nothing to define, but note that e' is new and, hence, closures of the form $(\_, e')$ do not appear in the range of m, respectively $m_1, \ldots, m_k$.

- If ψ'' is f_i^Q, then we have that $\psi''[\kappa] = \psi_i$. Moreover, we also have that $\mathsf{expand}(f_i^Q, e') = \mathsf{expand}(\mathsf{lookup}(f_i^Q, e')) = c_i$. Since m_i is a valid and good strategy mapping from ψ_i to c_i, we can use it do define m' via $m'(tu) = m_i(u)$ if u is in the domain of the syntax tree of ψ_i. Hence, m' is a valid strategy mapping on this subtree.

Clearly, m' is a valid strategy mapping from ψ' to $(\delta(Q), e')$. It is also good: On the subtrees of ψ' that are derived from substitution of one of the ψ_i, by definition m' agrees with m_i, which is good by assumption. On the parts of the tree derived from $\delta(Q)$, for all nodes in the syntax tree that are labeled by a fixpoint variable X, we have that X is of the form $X^{s'}_{Q',\mathcal{R}\upharpoonright Q'}$, which is unique for each $Q' \in \mathcal{Q}$. Hence, two nodes where m' yields (Q', e) must both be labeled by $X^{s'}_{Q',\mathcal{R}\upharpoonright Q'}$. Moreover, closures of this form do not appear in the range of the m_i, so m' is good. By the same reasoning, the pair of transitions from C to C' and from P to P' is also good.

□

Lemma 4.4.7. *Let $\mathcal{T}, v_I$ be a pointed LTS such that $\mathcal{T}, v_I \models \varphi_\mathcal{A}$. Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the acceptance game for $\mathcal{A}$ such that $\mathcal{V}$ plays with her strategy derived from the* HFL *model-checking game for $\varphi_\mathcal{A}$. Consider the unique infinite path in the unfolding tree of the play, and consider two distinct fixpoint variable configurations $C = (\_, (Q, \_), \_)$ and $C' = (\_, (Q', e), \_)$ such that C is the upper configuration and no other fixpoint variable configuration appears on the path between them. Let $P = (\_, X^s_{Q,\mathcal{R}})$ and $P' = (\_, X^{s'}_{Q',\mathcal{R}'})$ be the positions in the* HFL *model-checking game associated with C and C'. Then s' is descending from s with respect to $X_{Q,\mathcal{R}}$ and $\mathcal{R}' = \mathcal{R} \upharpoonright Q'$.*

Proof. Because C is a fixpoint configuration, the next configuration C'' will be of the form $(\_, (\delta(Q), e'), \_)$. In fact, $e = e'$, since the environment component from one configuration in an unfolding tree to the direct successor configuration changes only at fixpoint configurations. Since C' is the first fixpoint configuration after C, no other fixpoint configurations appear between C and C', whence it follows that $e = e'$. Moreover, Q' occurs in $\delta(Q)$. Let $P'' = (\_, \psi, \varepsilon)$ be the position in the HFL model-checking game for $\varphi_\mathcal{A}$ that is associated to C''. By Lemma 4.4.5, the occurrence of Q' in $\delta(Q)$ manifests itself in ψ via an occurrence of $X^{s''}_{Q',\mathcal{R}''}$ such that s'' is descending from s with respect to $X_{Q,\mathcal{R}}$. Note that, by Lemma 4.4.5, we have that $\mathcal{R}' = \mathcal{R} \upharpoonright Q'$.

Moreover, the strategy mapping for C'' and P'' maps $X^{s''}_{Q',\mathcal{R}}$ to (Q', e). Since $\mathcal{V}$ plays only good transitions, the strategy mapping for C' and P yields that $X^{s'}_{Q',\mathcal{R}'} = X^{s''}_{Q',\mathcal{R}''}$ since the image of $X^{s'}_{Q',\mathcal{R}'}$ under the strategy mapping is also (Q', e). Hence, $s' = s''$ and the lemma is proved. □

Lemma 4.4.8. *If $\mathcal{T}, v_I \models \varphi_\mathcal{A}$, then also $\mathcal{T}, v_I \models \mathcal{A}$.*

Proof. Consider an infinite play of the acceptance game for $\mathcal{A}$ such that $\mathcal{V}$ plays with her strategy derived from $\varphi_\mathcal{A}$. By Lemma 4.4.6, either $\mathcal{V}$ wins this play in finitely many steps, or the play is infinite. Consider the sequence of fixpoint configurations on the infinite path of the unfolding tree of such an infinite play. By Lemma 4.4.7, the sequence of associated positions in the HFL model-checking game is $(\_, X^{s_i}_{Q_i,\mathcal{R}_i}, \_)_{i\in\mathbb{N}}$

such that for all $i \in \mathbb{N}$, we have that s_{i+1} is descending from s_i with respect to $X_{Q_i,\mathcal{R}_i}$. Hence, this sequence of annotated fixpoint variables satisfies the conditions of Lemma 3.2.2, so there is $n \in \mathbb{N}$ and a greatest-fixpoint variable $X_{Q,\mathcal{R}}$ such that $Q_i, \mathcal{R}_i = Q, \mathcal{R}$ for infinitely many $i \geq n$ and $i \geq n$ implies that $X_{Q_i,\mathcal{R}_i} \prec X_{Q,\mathcal{R}}$ or $X_{Q_i,\mathcal{R}_i} = X_{Q,\mathcal{R}}$.

Now consider the sequence $(X_{Q_i,\mathcal{R}_i})_{i \geq n}$ of fixpoint variables. It remains to prove that no $X_{Q',\mathcal{R}'}$ with $\Delta(Q') > \Delta(Q)$ appears in that sequence. First we note that no variable $X_{Q',\mathcal{R}'}$ with $Q' \in \mathcal{R}$ and $\Delta(Q') > \Delta(Q)$ appears since in such a case necessarily $X_{Q',\mathcal{R}'} \succ X_{Q,\mathcal{R}}$. It remains to show that also no $X_{Q',\mathcal{R}'}$ with $\Delta(Q') > \Delta(Q)$ and $Q' \notin \mathcal{R}$ appears after n. For the sake of contradiction, assume that such a state appears in the sequence of fixpoint variables. Without loss of generality, let Q' be the largest such state with respect to $<_{\mathcal{Q}}$, and let the number $n' > n$ of its occurence be the smallest position in the sequence after n where Q' appears. Then, for all $i \geq n'$, we have that $Q' \in \mathcal{R}_i$. Clearly, this is true for $\mathcal{R}' = \mathcal{R}_{n'}$. Assume that we have shown that $Q' \in \mathcal{R}_i$ for some $i \geq n'$. By Lemma 4.4.7, we have that $\mathcal{R}_{i+1} = \mathcal{R}_i \upharpoonright Q_i$. If $Q_{i+1} \in \mathcal{R}_i$, note that, by assumption, $Q' \geq_{\mathcal{Q}} Q_{i+1}$ holds, whence $Q' \in \mathcal{R}_i \upharpoonright Q_{i+1}$ also holds. Similarly, if $Q_{i+1} \notin \mathcal{R}_i$, since $\Delta(Q') \geq \Delta(Q_{i+1})$, we also obtain $Q' \in \mathcal{R}_{i+1}$.

Since $Q' \notin \mathcal{R}$, we get that $X_{Q_i,\mathcal{R}_i} \neq X_{Q,\mathcal{R}}$ for any $i \geq n'$, which contradicts Lemma 3.2.2. It follows that the priority of Q is the highest priority of a fixpoint state that occurs infinitely often on the infinite path of the unfolding tree generated by the play in the APKA acceptance game. Since Q is a greatest-fixpoint state, $\mathcal{V}$ wins this play. □

Theorem 4.4.9. *Let $\mathcal{T}, v_I$ be a pointed LTS. Then $\mathcal{T}, v_I \models \mathcal{A}$ if and only if $\mathcal{T}, v_I \models \varphi_{\mathcal{A}}$.*

Proof. By Lemma 4.4.8, if $\mathcal{T}, v_I \models \varphi_{\mathcal{A}}$ then $\mathcal{T}, v_I \models \mathcal{A}$. On the other hand, if $\mathcal{T}, v_I \not\models \varphi_{\mathcal{A}}$, then by Lemma 3.1.4 we have $\mathcal{T}, v_I \models \overline{\varphi_{\mathcal{A}}}$. By Lemma 4.4.4, we have $\overline{\varphi_{\mathcal{A}}} = \varphi_{\overline{\mathcal{A}}}$, so by Lemma 4.2.22, we obtain that $\mathcal{T}, v_I \not\models \varphi_{\mathcal{A}}$ implies $\mathcal{T}, v_I \models \overline{\mathcal{A}}$ and hence $\mathcal{T}, v_I \not\models \mathcal{A}$. □

Remark 4.4.10. If $\mathcal{A}$ is an APKA with n states, then $|\varphi_{\mathcal{A}}|$ is at most exponential in n and polynomial in the combined size of the transition relation of $\mathcal{A}$. This is proved in Lemma 4.4.2. This blowup is in line with a similar blowup when translating PA to $\mathcal{L}_\mu$.

4.5 The Complexity of the Acceptance Game

We very briefly discuss the complexity of the APKA acceptance game. Note that, on any given class of structures, one can endow the fixpoint configurations in the acceptance game with counters, similar to μ-signatures, in order to make the game finite. Technically, this yields a model-checking algorithm for APKA and, hence for HFL. However, we have seen in Section 4.3, that for every HFL formula φ, there is an equivalent APKA $\mathcal{A}$ of the same type-theoretic order. Moreover, for each position in the model-checking game for φ, there is at least one position in the APKA acceptance game, linked to it via a strategy mapping. Considering the formulas from Section 3.3, it is immediate that the APKA acceptance game suffers from similar nonelementary blowup in the size of its game graph, since in this case, the counters associated with the fixpoint configurations can be taken directly from the μ-signatures in the corresponding HFL model-checking game. The reason for this blowup is the same as in the case of the HFL model-checking game: The environment structure used in APKA stores objects syntactically, not semantically, and this means that, given some semantic object, an APKA generally does not store this object as an optimally small representation. Hence, similarly to the HFL model-checking game, the APKA acceptance game should be considered a semantic tool. We use it in Chapter 6 to show that the fixpoint alternation hierarchy of a fragment of APKA, and, hence, of the corresponding fragment of HFL, is strict.

Chapter 5

Tail Recursion

This chapter contains the analysis of a fragment of HFL in which both boolean alternation and the interaction of higher-order behavior and extremal fixpoints are limited. This is inspired by the concept of *tail recursion* used in programming. It turns out that restricting HFL formulas to fixpoint recursion that is tail recursive lowers the complexity of the associated model-checking problem from k-EXPTIME to $(k-1)$-EXPSPACE if $k > 0$ is the type-theoretic order of the formula in question. We will see in Section 6.2.2 that one reason for this is that restriction to tail recursion forces the interaction between higher-order behavior and fixpoint recursion to degenerate almost completely to the behavior of $\mathcal{L}_\mu$, which is conceptually much simpler. The restrictions on boolean alternation parallel those in a fragment of $\mathcal{L}_\mu$ proposed in [35] and shown by the authors to be as expressive as ECTL* [89]. A similar concept is explored in the first-order setting in [11] under the name *solitaire games*.

Section 5.1 contains the definition of tail recursion in the context of HFL. Section 5.2 contains a top-down local model-checking procedure that works in the advertised space complexity. We provide a matching lower bound in Section 5.3 reducing the *order-k corridor-tiling problem* to the order-$(k+1)$ tail-recursive model-checking problem. The former is known to be a k-EXPSPACE-complete problem. The coding of the corridor tiling problem uses auxiliary encodings of large numbers into higher-order functions following a pattern proposed by Jones [49]. This chapter works completely on the formula side of the HFL/APKA pairing. We discuss similar restrictions to APKA in Section 6.2.2 under the name of *simple* APKA.

5.1 Definition of Tail-Recursive HFL

In the context of programming, tail recursion refers to recursive definitions of functions where recursive calls are the last action in a program routine. In this case, the return value of the definition is the return value of the recursive call. In particular, a tail-recursive function does not appear as an argument of a recursive call. This principle can also be used to describe the recursion exhibited by least and greatest fixpoints in HFL formulas. Originally, Lange and Lozes described tail recursion in the context of low orders of the polyadic version of HFL and used it to characterize PSPACE/$_\sim$, the class of all bisimulation-invariant PSPACE-queries [63]. The concept has since been extended to full (monadic) HFL in [21], while the requirements for an HFL formula to be tail recursive have been relaxed in a follow-up publication

[20]. For this thesis, the version from the latter publication is used. The presentation is based on it and follows it closely, in particular in this section and Section 5.2.

For the purposes of this chapter, we consider the connectives $\wedge, [a]$ to be native to HFL and not just syntactic sugar. For example, $\varphi_1 \wedge \varphi_2$ is not considered an abbreviation of $\neg(\neg\varphi_1 \vee \neg\varphi_2)$. Recall that the greatest-fixpoint quantifier $\nu X. X$ is native to HFL by definition.

Tail recursion is defined with respect to some order k, which usually is the order of the formula in question. Intuitively, order-k tail-recursion restricts the occurrence of free fixpoint variables in the syntax of formulas such that

- free fixpoint variables do not occur in an operand position,
- subformulas with free fixpoint variables can have fully unrestricted nondeterministic operators, i.e. $\vee, \langle a \rangle$, but limited universal branching, i.e. $\wedge, [a]$, or vice versa,
- negation is only allowed for fixpoint closed formulas,
- subformulas that are fixpoint closed and do not contain fixpoint binders of order k are not restricted in the position of free fixpoint variables.

In order to make the presentation more compact, we need a shorthand to denote that at least one set in a pair of sets is empty.

Definition 5.1.1. Let S_1, S_2 be sets. We write $S_1 \xleftrightarrow{\emptyset} S_2$ to denote that at least one of S_1 and S_2 is empty.

Note that, in particular, $\emptyset \xleftrightarrow{\emptyset} \emptyset$.

Definition 5.1.2. A fixpoint closed HFL formula φ of order k or less is called *order-k tail-recursive* if the statement $\mathsf{tail}^k(\varphi, \emptyset, A)$ for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$ can be derived via the rules in Figure 5.1. Note that, in this figure, we write _ for an unspecified element of $\{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$. We write $\mathrm{HFL}^k_{\mathsf{tail}}$ for the collection of order-k tail-recursive formulas in HFL^k.

Note that if $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ can be derived for some subformula ψ of φ, then the set $\mathcal{Y}$ is the set of free fixpoint variables of ψ. This follows by a straightforward induction from the fact that the only rule that adds variables to $\mathcal{Y}$ is the axiom for fixpoint variables, and the only rule to remove them is the rule for fixpoint binders. The three *modes* N, U and F indicate whether nondeterministic operators (N) or universal operators (U) can be used without restriction. For example, rules ($\vee$) and ($\vee_{\mathsf{U}}$) govern the behavior of tail recursion around disjunctions: If the subformula in question is in mode N, then both subformulas of a disjunction may contain free fixpoint variables, assuming a judgment for them in mode N can be derived. On the other hand, if a disjunction is in mode U, then at most one subformula can have free fixpoint variables. Note that, via rule (alter), subformulas without free fixpoint variables can always add modes N and U if there is a derivation for them for at least one mode.

Finally, a derivation for a subformula in mode F means that it does not contain fixpoint binders of order k. Intuitively this means that the formula is equivalent to one in HFL^{k-1} and, hence, harmless. For this reason, occurrences of free fixpoint

Figure 5.1: The derivation rules for order-k tail-recursion.

$$(\mathsf{alter})\ \frac{A \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi,\emptyset,\_)}{\mathsf{tail}^k(\varphi,\emptyset,A)}$$

$$(\mathsf{prop})\ \frac{}{\mathsf{tail}^k(P,\emptyset,\_)} \qquad (\mathsf{var})\ \frac{}{\mathsf{tail}^k(x,\emptyset,\_)} \qquad (\mathsf{fvar})\ \frac{}{\mathsf{tail}^k(X,\{X\},\_)}$$

$$(\neg)\ \frac{A, \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi,\emptyset,A)}{\mathsf{tail}^k(\neg\varphi,\emptyset,A)} \qquad (\neg_{\mathsf{F}})\ \frac{\mathsf{tail}^k(\varphi,\mathcal{Y},\mathsf{F})}{\mathsf{tail}^k(\neg\varphi,\mathcal{Y},\mathsf{F})}$$

$$(\vee)\ \frac{A \in \{\mathsf{N},\mathsf{F}\} \qquad \mathsf{tail}^k(\varphi_1,\mathcal{Y}_1,A) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{Y}_2,A)}{\mathsf{tail}^k(\varphi \vee \varphi_2,\mathcal{Y}_1 \cup \mathcal{Y}_2,A)}$$

$$(\vee_{\mathsf{U}})\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{Y}_1,\mathsf{U}) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{Y}_2,\mathsf{U}) \qquad \mathcal{Y}_1 \overset{\emptyset}{\longleftrightarrow} \mathcal{Y}_2}{\mathsf{tail}^k(\varphi \vee \varphi_2,\mathcal{Y}_1 \cup \mathcal{Y}_2,\mathsf{U})}$$

$$(\wedge)\ \frac{A \in \{\mathsf{U},\mathsf{F}\} \qquad \mathsf{tail}^k(\varphi_1,\mathcal{Y}_1,A) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{Y}_2,A)}{\mathsf{tail}^k(\varphi \wedge \varphi_2,\mathcal{Y}_1 \cup \mathcal{Y}_2,A)}$$

$$(\wedge_{\mathsf{N}})\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{Y}_1,\mathsf{N}) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{Y}_2,\mathsf{N}) \qquad \mathcal{Y}_1 \overset{\emptyset}{\longleftrightarrow} \mathcal{Y}_2}{\mathsf{tail}^k(\varphi \wedge \varphi_2,\mathcal{Y}_1 \cup \mathcal{Y}_2,\mathsf{N})}$$

$$(\langle a\rangle)\ \frac{A \in \{\mathsf{N},\mathsf{F}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{Y},A)}{\mathsf{tail}^k(\langle a\rangle\varphi,\mathcal{Y},A)} \qquad (\langle a\rangle_{\mathsf{U}})\ \frac{\mathsf{tail}^k(\varphi,\emptyset,\mathsf{U})}{\mathsf{tail}^k(\langle a\rangle\varphi,\emptyset,\mathsf{U})}$$

$$([a])\ \frac{A \in \{\mathsf{U},\mathsf{F}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{Y},A)}{\mathsf{tail}^k([a]\varphi,\mathcal{Y},A)} \qquad ([a]_{\mathsf{N}})\ \frac{\mathsf{tail}^k(\varphi,\emptyset,\mathsf{N})}{\mathsf{tail}^k([a]\varphi,\emptyset,\mathsf{N})}$$

$$(\mathsf{app})\ \frac{A \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi_1,\mathcal{Y},A) \qquad \mathsf{tail}^k(\varphi_2,\emptyset,\_)}{\mathsf{tail}^k(\varphi_1\,\varphi_2,\mathcal{Y},A)}$$

$$(\mathsf{app}_{\mathsf{F}})\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{Y}_1,\mathsf{F}) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{Y}_2,\mathsf{F})}{\mathsf{tail}^k(\varphi_1\,\varphi_2,\mathcal{Y}_1 \cup \mathcal{Y}_2,\mathsf{F})} \qquad (\lambda)\ \frac{A \in \{\mathsf{N},\mathsf{U},\mathsf{F}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{Y},A)}{\mathsf{tail}^k(\lambda(x^v : \tau).\,\varphi,\mathcal{Y},A)}$$

$$(\mathsf{fp})\ \frac{\sigma \in \{\mu,\nu\} \qquad A \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{Y},A)}{\mathsf{tail}^k(\sigma(X : \tau).\,\varphi,\mathcal{Y} \setminus \{X\},A)}$$

$$(\mathsf{fp}_{\mathsf{F}})\ \frac{\sigma \in \{\mu,\nu\} \qquad ord(\tau) < k \qquad \mathsf{tail}^k(\varphi,\mathcal{Y},\mathsf{F})}{\mathsf{tail}^k(\sigma(X : \tau).\,\varphi,\mathcal{Y} \setminus \{X\},\mathsf{F})}$$

variables are completely unrestricted in a subformula that has a derivation for mode F. This includes fixpoint variables in operand position. However, note that a derivation for mode F is only useful if it ends in some subformula that is fixpoint-variable closed.

Example 5.1.3. Consider the HFL formula φ from Example 2.4.6 defined as

$$\Big(\mu(F\colon \tau \to \bullet).\, \lambda(g\colon \tau).\, g\, P \vee \big(F\, \lambda(y\colon \bullet).\, g\,(g\, y)\big)\Big)(\lambda(z\colon \bullet).\, \langle a\rangle z)$$

where $\tau = \bullet \to \bullet$. It is order-2 tail-recursive. In fact, even the formula

$$\Big(\mu(F\colon \tau \to \bullet).\, \lambda(g\colon \tau).\, g\, P \vee \big(F\, \lambda(y\colon \bullet).\, g\,(g\, y)\big)\Big)(\mu(X\colon \bullet \to \bullet).\, \lambda(z\colon \bullet).\, [a]z)$$

is tail recursive. The fixpoint formula of X does occur in an operand position, but it has no free fixpoint variables.

From the definition of order-0-tail recursion, it is not hard to see that $\mathrm{HFL}^0_{\mathsf{tail}}$ coincides with the logic L_2 from [35].

Definition 5.1.4. A fixpoint formula φ is called *strictly tail-recursive* if the statement $\mathsf{tail}^k(\varphi, \emptyset, A)$ for $A \in \{\mathsf{N}, \mathsf{U}\}$ can be derived via the rules in Figure 5.1 without using rule (fp_F). In this case, we write $\mathsf{tail}^\mathsf{s}(\varphi, \emptyset, A)$. We write $\mathrm{HFL}^k_{\mathsf{s\text{-}tail}}$ for the collection of strictly tail-recursive formulas in HFL^k.

Note that strict tail-recursion is not defined for each order since the only rule referring to type order does not occur in any derivation of strict tail-recursiveness. However, any formula in $\mathrm{HFL}^k_{\mathsf{s\text{-}tail}}$ is also in $\mathrm{HFL}^k_{\mathsf{tail}}$. Note that strict tail-recursion is still more relaxed than the version of tail recursion defined in [21] since in strict tail-recursion, switching between modes in a formula is still allowed.

Example 5.1.5. Recall that $\tau = \bullet \to \bullet$. The HFL-formulas from Example 2.4.6, respectively Example 5.1.3 are strictly tail recursive. The formula

$$\Big(\mu(F\colon \tau \to \bullet).\, \lambda(g\colon \tau).\, g\, P \vee \big(F\, \lambda(y\colon \bullet).\, g\,(g\, y)\big)\Big)(\mu(X\colon \tau).\, \lambda(z\colon \bullet).\, [a](z \vee X))$$

is order-2 tail-recursive, but not strictly tail-recursive, since the argument-side formula $\mu(X\colon \tau).\, \lambda(z\colon \bullet).\, [a](z \vee X)$ in the operand is not tail recursive due to the occurrence of X under both $\vee$ and a modal box.

5.2 Upper Bounds for Model-Checking

The model-checking algorithm presented in this section requires some preprocessing. For example, before running the algorithm, information has to be collected on the exact derivation tree that proves tail recursiveness of a given formula. This is necessary for example in order to know which of the two subformulas of, e.g., a disjunction contains free fixpoint variables in a setting where only one is allowed to do so.

Algorithm 1 Checking for order-k tail-recursion.

```
1: procedure VerTR($\varphi, k$)                                   ▷ Returns ($\mathcal{Y}$, $modes$)
2:     switch $\varphi$ do
3:         case $\varphi = P$ return $(\emptyset, \{\mathsf{N}, \mathsf{U}, \mathsf{F}\})$
4:         case $\varphi = X$ return $(\{X\}, \{\mathsf{N}, \mathsf{U}, \mathsf{F}\})$
5:         case $\varphi = x$ return $(\emptyset, \{\mathsf{N}, \mathsf{U}, \mathsf{F}\})$
6:         case $\varphi = \varphi_1 \vee \varphi_2$
7:             $(\mathcal{Y}_i, modes_i) \leftarrow$ ModeTC(VerTR($\varphi_i, k$)), $i \in \{1, 2\}$
8:             if $\mathcal{Y}_1 \overset{\emptyset}{\longleftrightarrow} \mathcal{Y}_2$ then
9:                 return $(\mathcal{Y}_1 \cup \mathcal{Y}_2, modes_1 \cap modes_2)$
10:            else return $(\mathcal{Y}_1 \cup \mathcal{Y}_2, (modes_1 \cap modes_2) \setminus \{\mathsf{U}\})$
11:        case $\varphi = \varphi_1 \wedge \varphi_2$
12:            $(\mathcal{Y}_i, modes_i) \leftarrow$ ModeTC(VerTR($\varphi_i, k$)), $i \in \{1, 2\}$
13:            if $\mathcal{Y}_1 \overset{\emptyset}{\longleftrightarrow} \mathcal{Y}_2$ then
14:                return $(\mathcal{Y}_1 \cup \mathcal{Y}_2, modes_1 \cap modes_2)$
15:            else return $(\mathcal{Y}_1 \cup \mathcal{Y}_2, (modes_1 \cap modes_2) \setminus \{\mathsf{N}\})$
16:        case $\varphi = \langle a \rangle \varphi'$
17:            $(\mathcal{Y}, modes) \leftarrow$ ModeTC(VerTR($\varphi', k$))
18:            if $\mathcal{Y} = \emptyset$ then
19:                return $(\mathcal{Y}, modes)$
20:            else return $(\mathcal{Y}, modes \setminus \{\mathsf{U}\})$
21:        case $\varphi = [a]\varphi'$
22:            $(\mathcal{Y}, modes) \leftarrow$ ModeTC(VerTR($\varphi', k$))
23:            if $\mathcal{Y} = \emptyset$ then
24:                return $(\mathcal{Y}, modes)$
25:            else return $(\mathcal{Y}, modes \setminus \{\mathsf{N}\})$
26:        case $\varphi = \neg\varphi'$
27:            $(\mathcal{Y}, modes) \leftarrow$ ModeTC(VerTR($\varphi', k$))
28:            if $\mathcal{Y} = \emptyset$ then
29:                return $(\mathcal{Y}, modes)$
30:            else return $(\mathcal{Y}, modes \cap \{\mathsf{F}\})$
31:        case $\varphi = \lambda x.\, \varphi'$ return VerTR($\varphi', k$)
32:        case $\varphi = \varphi_1\, \varphi_2$
33:            $(\mathcal{Y}_i, modes_i) \leftarrow$ ModeTC(VerTR($\varphi_i, k$)), $i \in \{1, 2\}$
34:            if $\mathcal{Y}_2 = \emptyset$ and $modes_2 \neq \emptyset$ then
35:                return $(\mathcal{Y}_1, modes_1)$
36:            else return $(\mathcal{Y}_1 \cup \mathcal{Y}_2, modes_1 \cap modes_2 \cap \{\mathsf{F}\})$
37:        case $\varphi = \sigma(X : \tau).\, \varphi'$
38:            $(\mathcal{Y}, modes) \leftarrow$ ModeTC(VerTR($\varphi, k$))
39:            if $ord(\tau) < k$ then
40:                return $(\mathcal{Y} \setminus \{X\}, modes)$
41:            else return $(\mathcal{Y} \setminus \{X\}, modes \setminus \{\mathsf{F}\})$
42: procedure ModeTC(($\mathcal{Y}$, $modes$))
43:     if $\mathcal{Y} = \emptyset$ and $modes \neq \emptyset$ then
44:         return $(\mathcal{Y}, modes \cup \{\mathsf{N}, \mathsf{U}\})$
45:     else return $(\mathcal{Y}, modes)$
```

5.2.1 Verifying Tail Recursiveness

In preparation for the upper bound result, we first show that it can be verified in time linear in the size of the syntax tree of a formula whether it is order-k tail-recursive. For this we use a bottom-up procedure VerTR that collects all those modes such that a derivation in the respective mode is possible for the given subformula. This does not mean that the derivation can be extended to one for the full formula. Intuitively, this is the same principle as the powerset construction used to determinize finite automata, although it only operates on the set $\mathit{modes} = \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$. The procedure ModeTC that is used in the definition of VerTR emulates the derivation rule (alter), i.e., it adds modes N and U to subformulas that are fixpoint closed and have at least one successful derivation.

Lemma 5.2.1. *Let φ be an* HFL *formula. Then $\mathsf{tail}^k(\varphi, \mathcal{Y}, A)$ is derivable for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$ if and only if $\mathsf{VerTR}(\varphi, k) = (\mathcal{Y}, \mathit{modes})$ and $A \in \mathit{modes}$. Hence, it is decidable in $\mathcal{O}(|\varphi|)$ whether φ is order-k tail-recursive.*

Proof. Procedure VerTR in Algorithm 1 checks whether an HFL formula φ is order-k tail-recursive. We prove by induction over the syntax tree of φ that $\mathsf{tail}^k(\varphi, \mathcal{Y}, A)$ for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$ is derivable if and only if $\mathsf{ModeTC}(\mathsf{VerTR}(\varphi, k))$ returns $(\mathcal{Y}, \mathit{modes})$ and $A \in \mathit{modes}$. Let ψ be a subformula of φ and assume that the statement has been proved for all proper subformulas of ψ. Depending on the form of ψ, the argument proceeds as follows. The cases of ψ being of the form P, X and x are immediate.

- If $\psi = \psi_1 \vee \psi_2$, let $(\mathcal{Y}_i, \mathit{modes}_i)$ for $i \in \{1, 2\}$ be the return value of the call to $\mathsf{ModeTC}(\mathsf{VerTR}(\varphi_i, k))$ for $i \in \{1, 2\}$. By the induction hypothesis, we have that if $A \in \mathit{modes}_i$, then $\mathsf{tail}^k(\varphi_i, \mathcal{Y}_i, A)$ is derivable. Note that the only two applicable rules are ($\vee$) and ($\vee_{\mathsf{U}}$). If $\mathsf{N} \in \mathit{modes}_1$ and $\mathsf{N} \in \mathit{modes}_2$, then $\mathsf{tail}^k(\varphi, \mathcal{Y}_1 \cup \mathcal{Y}_2, \mathsf{N})$ is derivable, and the same holds for F. In both cases, both return statements of $\mathsf{VerTR}(\psi, k)$ return $(\mathcal{Y}_1 \cup \mathcal{Y}_2, \mathit{modes})$ with $\mathsf{N} \in \mathit{modes}$, respectively $\mathsf{F} \in \mathit{modes}$ since $\mathit{modes}_1 \cap \mathit{modes}_2$ contains N, respectively F.

  Moreover, if at least one of the $\mathcal{Y}_i$ is $\emptyset$, then rule ($\vee_{\mathsf{U}}$) is potentially applicable. If $\mathcal{Y}_i$ is $\emptyset$ for both $i = 1$ and $i = 2$, then $\mathit{modes}_1 \cap \mathit{modes}_2$ contains U and rule ($\vee_{\mathsf{U}}$) allows us to derive $\mathsf{tail}^k(\psi, \emptyset, \mathsf{U})$, and also $\mathsf{VerTR}(\psi, k)$ returns $(\emptyset, \mathit{modes})$ with $\mathsf{U} \in \mathit{modes}$. If there is $j \in \{1, 2\}$ such that $\mathcal{Y}_j = \emptyset$ but $\mathcal{Y}_{1-j} \neq \emptyset$, then rule ($\vee_{\mathsf{U}}$) is applicable if and only if modes_{1-j} contains U and $\mathit{modes}_j \neq \emptyset$. In this case rule ($\mathsf{alter}$) implies $\mathsf{U} \in \mathit{modes}_j$ and rule ($\vee_{\mathsf{U}}$) allows us to derive $\mathsf{tail}^k(\psi, \mathcal{Y}_1 \cup \mathcal{Y}_2, \mathsf{U})$, and in this case, $\mathsf{VerTR}(\psi, k)$ returns $(\mathcal{Y}_1 \cup \mathcal{Y}_2, \mathit{modes})$ with $\mathsf{U} \in \mathit{modes}$.

- If $\psi = \psi_1 \wedge \psi_2$, the argument proceeds completely symmetrically to that for the case $\psi = \psi_1 \vee \psi_2$.

- If $\psi = \langle a \rangle \psi'$, let $(\mathcal{Y}, \mathit{modes}')$ be the return value of $\mathsf{ModeTC}(\mathsf{VerTR}(\psi', k))$. By the induction hypothesis, we have that if $A \in \mathit{modes}'$ then $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable. Note that the only two applicable rules are ($\langle a \rangle$) and ($\langle a \rangle_{\mathsf{U}}$). For $A = \mathsf{N}$ or $A = \mathsf{F}$, rule ($\langle a \rangle$) yields that $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable if and only if $\mathsf{tail}^k(\langle a \rangle \psi', \mathcal{Y}, A)$ is derivable, and both return statements of $\mathsf{VerTR}(\psi, k)$ yield $(\mathcal{Y}, \mathit{modes})$ with $A \in \mathit{modes}$ if $A \in \mathit{modes}'$.

  Moreover, if $\mathcal{Y} = \emptyset$, rule ($\langle a \rangle_{\mathsf{U}}$) is applicable and $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{U})$ is derivable via this rule if also $\mathsf{U} \in \mathit{modes}'$. Only in this case the return value of $\mathsf{VerTR}(\psi, k)$

is obtained via the first return statement, whence said return value contains U if and only if modes' contains U.

- If $\psi = [a]\psi'$, the argument proceeds completely symmetrically to the case that $\psi = \langle a\rangle\psi'$.

- If $\psi = \neg\psi'$, let $(\mathcal{Y}, \mathit{modes}')$ be the return value of $\mathsf{ModeTC}(\mathsf{VerTR}(\psi', k))$. By the induction hypothesis, we have that if $A \in \mathit{modes}'$ then $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable. Note that the only two applicable rules are $(\neg)$ and $(\neg_{\mathsf{F}})$. If $\mathcal{Y} = \emptyset$, then $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ is derivable via rule $(\neg)$ for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$ if and only if $\mathsf{tail}^k(\psi', \mathcal{Y}, k)$ is derivable, and $\mathsf{VerTR}(\psi, k)$ returns $(\mathcal{Y}, \mathit{modes}')$. However, if $\mathcal{Y} \neq \emptyset$, then only rule $(\neg_{\mathsf{F}})$ is applicable and $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ is derivable via rule $(\neg_{\mathsf{F}})$ if and only if $A = \mathsf{F}$ and $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable. The return value of $\mathsf{VerTR}(\psi, k)$ is then determined via the second return call and returns $(\mathcal{Y}, \mathit{modes})$ with $\mathit{modes} = \{\mathsf{F}\}$ if and only if $\mathsf{F} \in \mathit{modes}'$.

- If $\psi = \lambda x.\, \psi'$, let $(\mathcal{Y}, \mathit{modes}')$ be the return value of $\mathsf{ModeTC}(\mathsf{VerTR}(\psi', k))$ and also the return value of $\mathsf{ModeTC}(\mathsf{VerTR}(\psi, k))$. On the other hand, we have that $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ is derivable via the only applicable rule (λ) if and only if $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable. Since, by the induction hypothesis, we have that if $A \in \mathit{modes}'$ then $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$, the algorithm works correctly for this case.

- If $\psi = \psi_1\, \psi_2$, let the return value of $\mathsf{ModeTC}(\mathsf{VerTR}(\varphi_i, k))$ be $(\mathcal{Y}_i, \mathit{modes}_i)$ for $i \in \{1, 2\}$. By the induction hypothesis, we have that if $A \in \mathit{modes}_i$, then $\mathsf{tail}^k(\varphi_i, \mathcal{Y}_i, A)$ is derivable. Note that the only applicable rules are (app) and $(\mathsf{app}_{\mathsf{F}})$. Rule (app) is only applicable if $\mathcal{Y}_2 = \emptyset$, in which case $\mathsf{tail}^k(\psi, \mathcal{Y}_1, A)$ is derivable if $A \in \mathit{modes}_1$ and $\mathit{modes}_2 \neq \emptyset$. In this case, the return value of $\mathsf{VerTR}(\psi, k)$ is obtained via the first return call and is $(\mathcal{Y}_1, \mathit{modes})$ with $A \in \mathit{modes}$ if and only if $A \in \mathit{modes}_1$. Note that, in particular, this contains the case that $\mathit{modes}_1 \cap \mathit{modes}_2$ contains F, in which case also rule $(\mathsf{app}_{\mathsf{F}})$ would be applicable to derive $\mathsf{tail}^k(\psi, \mathcal{Y}_1, \mathsf{F})$.

  If $\mathcal{X}_2 \neq \emptyset$, then only rule $(\mathsf{app}_{\mathsf{F}})$ is applicable and only $\mathsf{tail}^k(\psi, \mathcal{Y}_1 \cup \mathcal{Y}_2, \mathsf{F})$ is derivable if $\mathsf{F} \in \mathit{modes}_1$ and $\mathsf{F} \in \mathit{modes}_2$. In this case, the return value of $\mathsf{VerTR}(\psi, k)$ is obtained via the second return call and is $(\mathcal{Y}_1 \cup \mathcal{Y}_2, \{\mathsf{F}\})$ if and only if $\mathsf{F} \in \mathit{modes}_1 \cap \mathit{modes}_2$.

- If $\psi = \sigma(X : \tau).\, \psi'$, then let the return value of $\mathsf{ModeTC}(\mathsf{VerTR}(\psi', k))$ be $(\mathcal{Y}, \mathit{modes}')$. By the induction hypothesis, we have that if $A \in \mathit{modes}'$ then $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable. Note that only rules (fp) and $(\mathsf{fp}_{\mathsf{F}})$ are applicable. If $\mathit{ord}(\tau) < k$ then $\mathsf{tail}^k(\psi, \mathcal{Y} \setminus \{X\}, A)$ is derivable via rule $(\mathsf{fp}_{\mathsf{F}})$ if and only if $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable. Procedure $\mathsf{VerTR}(\psi, k)$ mirrors this in its first return call by returning $(\mathcal{Y} \setminus \{X\}, \mathit{modes}')$.

  However, if $\mathit{ord}(\tau) = k$ then only rule (fp) is applicable, and $\mathsf{tail}^k(\psi, \mathcal{Y} \setminus \{X\}, A)$ is derivable via rule (fp) if and only if $\mathsf{tail}^k(\psi', \mathcal{Y}, A)$ is derivable and $A \neq \mathsf{F}$. Procedure $\mathsf{VerTR}(\psi, k)$ mirrors this in its first return call by returning $(\mathcal{Y} \setminus \{X\}, \mathit{modes}' \setminus \{\mathsf{F}\})$.

By applying the result of the induction to (φ, k), we obtain that $\mathsf{tail}^k(\varphi, \emptyset, A)$ is derivable for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$ if and only if $\mathsf{VerTR}(\varphi, k) = (\emptyset, \mathit{modes})$ with $A \in \mathit{modes}$. For the complexity results, note that procedure VerTR does exactly one recursive call per subformula of φ and calls ModeTC at most once per subformula. The latter procedure runs in constant time, while the former procedure has a constant inner loop. Hence, the overall procedure runs in time in $\mathcal{O}(|\varphi|)$. $\square$

Remark 5.2.2. Algorithm 1 can be adapted to verify whether a formula is strictly tail recursive by replacing the entire if-construct in the case for fixpoint bindings by its else-clause, i.e., by always returning $(\mathcal{Y}, \mathit{modes} \setminus \{\mathsf{F}\})$. Since this is the only place where rule $(\mathsf{fp_F})$ is invoked in the correctness proof, it is not hard to see that $\mathsf{tail}^{\mathsf{s}}(\varphi, \mathcal{Y}, A)$ is derivable if and only if the modified algorithm returns $(\mathcal{Y}, \mathit{modes})$ with $A \in \mathit{modes}$.

While Algorithm 1 yields a procedure to decide whether a given formula is order-k tail-recursive, it does not establish the concrete derivation tree. However, with an additional top-down procedure, this tree can be constructed. In order to construct this tree, we generate, for each formula ψ in the syntax *tree* of φ, a triple $\mathsf{info}(\psi) = (\mathcal{Y}, A, A')$ with the following intuition: $\mathcal{Y}$ contains the free fixpoint variables of ψ for notational convenience, while A and A' signal which facts about ψ are used in the derivation of order-k tail-recursiveness of φ. Each subformula, with the exception of φ itself, variable formulas, atomic formulas and another exception to be made clear, appears exactly twice in the derivation tree for order-k tail-recursiveness of φ, once in the premises of a derivation rule, and once in the conclusion. In the case that $\mathsf{info}(\psi) = (\mathcal{Y}, A, A')$, the fact that $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ is derivable is used as a premise of a derivation rule, while the fact that $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ is derivable is used in the conclusion of a derivation rule. For example, if the triples $\mathsf{info}(\psi_1 \vee \psi_2) = (\mathcal{Y}, \_, \mathsf{U})$, $\mathsf{info}(\psi_1) = (\mathcal{Y}, \mathsf{U}, \_)$ and $\mathsf{info}(\psi_2) = (\emptyset, \mathsf{U}, \_)$, are generated, then we can conclude that the following instance of rule $(\vee_\mathsf{U})$ was used to derive order-k tail-recursiveness of φ:

$$(\vee_\mathsf{U})\ \frac{\mathsf{tail}^k(\psi_1, \mathcal{Y}, \mathsf{U}) \qquad \mathsf{tail}^k(\psi_2, \emptyset, \mathsf{U}) \qquad \mathcal{Y} \xleftrightarrow{\emptyset} \emptyset}{\mathsf{tail}^k(\psi_1 \vee \psi_2, \mathcal{Y} \cup \emptyset, \mathsf{U})}$$

Note that $A \neq A'$ can only occur if $\mathcal{Y} = \emptyset$. This also signals that an application of rule (alter) was used on ψ. This is the exception to the rule that each subformula that is not φ itself, a variable formula, or atomic occurs exactly twice in the derivation tree. During the procedure that generates the triples, we also allow A and A' to be temporarily set to ε, which signals that the final value is not decided yet.

Begin the procedure with the tuple $\mathsf{info}(\varphi) = (\mathcal{Y}, A, \varepsilon)$ such that $A \in \mathit{modes}$, where $(\mathcal{Y}, \mathit{modes}) = \mathsf{ModeTC}(\mathsf{VerTR}(\varphi, k))$. If given a tuple of the form $\mathsf{info}(\psi) = (\mathcal{Y}, A, \varepsilon)$ for some subformula ψ of φ, by assumption $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ is derivable. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A')$ with A' depending on the form of ψ as follows:

Case ψ is P or X or x. Then ψ is a leaf formula. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A)$. Note that the axiom rules (prop), (var) and (fvar) are applicable with premise A.

Case ψ is of the form $\neg\psi'$, $\langle a\rangle\psi'$, $[a]\psi'$ or $\lambda x.\ \psi'$. In all of these cases, let $(\mathcal{Y}, \mathit{modes}) = \mathsf{ModeTC}(\mathsf{VerTR}(\psi, k))$ and let $(\mathcal{Y}', \mathit{modes}') = \mathsf{ModeTC}(\mathsf{VerTR}(\psi', k))$. Note that necessarily $\mathcal{Y} = \mathcal{Y}'$ since both formulas have the same free fixpoint variables, and that $\mathit{modes} \subseteq \mathit{modes}'$: If $A' \in \mathit{modes}$ for some $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{F}\}$, then $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ is derivable from $\mathsf{tail}^k(\psi', \mathcal{Y}', A')$ via one of the rules $(\neg)$, $(\neg_\mathsf{F})$, $(\langle a\rangle)$, $(\langle a\rangle_\mathsf{U})$, $([a])$, $([a]_\mathsf{N})$ or (λ), or $\mathcal{Y} = \mathcal{Y}' = \emptyset$, whence there is $A'' \in \mathit{modes}'$ such that $\mathsf{tail}^k(\psi, \mathcal{Y}, A'')$ is derivable from $\mathsf{tail}^k(\psi', \mathcal{Y}', A'')$ via one of the above rules, and $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ is derivable from $\mathsf{tail}^k(\psi, \mathcal{Y}, A'')$ via rule (alter). In this case, rule (alter)

is also applicable to $\mathsf{tail}^k(\psi', \mathcal{Y}', A'')$ whence $A' \in \mathit{modes}'$. Set $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A)$ and continue with $\mathsf{info}(\psi') = (\mathcal{Y}', A, \varepsilon)$.

Case ψ is of the form $\sigma(X\colon \tau).\, \psi'$. Let $(\mathcal{Y}, \mathit{modes}) = \mathsf{ModeTC}(\mathsf{VerTR}(\psi, k))$ and let $(\mathcal{Y}', \mathit{modes}') = \mathsf{ModeTC}(\mathsf{VerTR}(\psi', k))$. Note that $\mathcal{Y}' = \mathcal{Y} \cup \{X\}$ or $\mathcal{Y}' = \mathcal{Y}$. There are two cases: If $\mathit{ord}(\tau) < k$, then $\mathit{modes}' \subseteq \mathit{modes}'$ since, if $A' \in \mathit{modes}'$, rules (fp) and $(\mathsf{fp_F})$ allow $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ to be derived from $\mathsf{tail}^k(\psi', \mathcal{Y}', A')$. If $A \in \mathit{modes}'$, update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A)$ and continue with $\mathsf{info}(\psi') = (\mathcal{Y}', A, \varepsilon)$. If $A \notin \mathit{modes}'$, there must be $A' \in \mathit{modes}'$ with $A' \neq A$ for if $\mathit{modes}' = \emptyset$, also $\mathit{modes} = \emptyset$ since the only applicable rules with conclusion $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ are (fp), $(\mathsf{fp_F})$ and (alter). The first two are not applicable if $\mathit{modes}' = \emptyset$, and rule (alter) requires a premise of the form $\mathsf{tail}^k(\psi, \mathcal{Y}, A'')$ with $A'' \neq A'$ and that premise must necessarily be derived via a rule different from (alter), which does not exist. Hence, there is $A' \in \mathit{modes}'$ with $A' \neq A$. By the same reasoning, $\mathcal{Y} = \emptyset$. Since also $A' \in \mathit{modes}$, the premise $\mathsf{tail}^k(\psi', \mathcal{Y}', A')$, which is derivable by the definition of $\mathsf{ModeTC}(\mathsf{VerTR})$, allows $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ to be derived via rule (fp) or $(\mathsf{fp_F})$. Since $\mathcal{Y} = \emptyset$, rule (alter) allows $\mathsf{tail}^k(\psi, \emptyset, A)$ to be derived. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A')$ and continue with $\mathsf{info}(\psi') = (\mathcal{Y}', A', \varepsilon)$.

The second case is that $\mathit{ord}(\tau) = k$. Note that if $A' \in \{\mathsf{N}, \mathsf{U}\} \cap \mathit{modes}'$, then $A' \in \mathit{modes}$ since if $A' \in \mathit{modes}'$ then rule (fp) allows $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ to be derived from $\mathsf{tail}^k(\psi', \mathcal{Y}', A')$. If $A \in \mathit{modes}'$, update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A)$ and continue with $\mathsf{info}(\psi') = (\mathcal{Y}', A, \varepsilon)$. If $A \notin \mathit{modes}'$, via reasoning similar to the case of $\mathit{ord}(\tau) < k$ we obtain that $\mathit{modes} = \emptyset$ and there is $A' \in \mathit{modes}' \setminus \{\mathsf{F}\}$ with $A' \neq A$. Then also $A' \in \mathit{modes}$ and the premise $\mathsf{tail}^k(\psi', \mathcal{Y}', A')$, which is derivable by the definition of $\mathsf{ModeTC}(\mathsf{VerTR})$, allows $\mathsf{tail}^k(\psi, \mathcal{Y}, A')$ to be derived via rule (fp). Since $\mathcal{Y} = \emptyset$, rule (alter) allows $\mathsf{tail}^k(\psi, \emptyset, A)$ to be derived. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A')$ and continue with $\mathsf{info}(\psi') = (\mathcal{Y}', A', \varepsilon)$.

Case ψ is of the form $\psi_1\, \psi_2$. Let $(\mathcal{Y}, \mathit{modes}) = \mathsf{ModeTC}(\mathsf{VerTR}(\psi, k))$ and let $(\mathcal{Y}_i, \mathit{modes}_i) = \mathsf{ModeTC}(\mathsf{VerTR}(\psi_i))$ for $i \in \{1, 2\}$. If $A = \mathsf{F}$, then $\mathsf{tail}^k(\psi_i, \mathcal{Y}_i, \mathsf{F})$ is derivable for $i \in \{1, 2\}$ since the only rule with the conclusion $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{F})$ is rule $(\mathsf{app_F})$ which has premises $\mathsf{tail}^k(\psi_i, \mathcal{Y}_i, \mathsf{F})$ for $i \in \{1, 2\}$. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, \mathsf{F}, \mathsf{F})$ and continue with both $\mathsf{info}(\psi_1) = (\mathcal{Y}_1, \mathsf{F}, \varepsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{Y}_2, \mathsf{F}, \varepsilon)$.

If $A \neq \mathsf{F}$, then there are two cases: If $\mathcal{Y} = \emptyset$, then also $\mathcal{Y}_i = \emptyset$ for $i \in \{1, 2\}$. By the same reasoning as in the case for negation, modal operators, etc., we have that A is also in modes_i for $i \in \{1, 2\}$, whence rule (app) is applicable with premises $\mathsf{tail}^k(\psi_1, \emptyset, A)$ and $\mathsf{tail}^k(\psi_2, \emptyset, A)$. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A)$ and continue with both $\mathsf{info}(\psi_i) = (\mathcal{Y}_1, A, \varepsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{Y}, A, \varepsilon)$. If $\mathcal{Y} \neq \emptyset$, note that there is no premise such that the conclusion of (alter) yields $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$, whence $\mathsf{tail}^k(\psi, \mathcal{Y}, A)$ is derived from (app). Hence, $\mathcal{Y}_1 = \mathcal{Y}$ since $\mathcal{Y}_2 = \emptyset$ for otherwise rule (app) would not be applicable. It follows that $A \in \mathit{modes}_1$ and that $\mathit{modes}_2 \neq \emptyset$, whence rule (app) is applicable with premises $\mathsf{tail}^k(\psi_1, \mathcal{Y}_1, A)$ and $\mathsf{tail}^k(\psi_2, \emptyset, A')$ with $A' \in \mathit{modes}$. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, A, A)$ and continue with $\mathsf{info}(\psi_1) = (\mathcal{Y}_1, A, \varepsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{Y}_2, A', \varepsilon)$.

Case ψ is of the form $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$. Let $(\mathcal{Y}, \mathit{modes}) = \mathsf{ModeTC}(\mathsf{VerTR}(\psi, k))$ and let $(\mathcal{Y}_i, \mathit{modes}_i) = \mathsf{ModeTC}(\mathsf{VerTR}(\psi_i))$ for $i \in \{1, 2\}$. Without loss of generality, $\psi = \psi_1 \vee \psi_2$, the case for $\wedge$ is completely symmetric. If $A = \mathsf{F}$, then $\mathsf{F} \in \mathit{modes}_i$ for $i \in \{1, 2\}$ since the only rule with conclusion $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{F})$ is rule $(\vee)$ with premises $\mathsf{tail}^k(\psi_i, \mathcal{Y}_i, \mathsf{F})$ for $\in \{1, 2\}$. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, \mathsf{F}, \mathsf{F})$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{Y}_i, \mathsf{F}, \varepsilon)$ for $i \in \{1, 2\}$.

If $A = \mathsf{N}$, then $\mathsf{N} \in \mathit{modes}_i$ for $i \in \{1,2\}$. For the sake of contradiction, assume that $\mathsf{N} \notin \mathit{modes}_i$ for some $i \in \{1,2\}$. Then $\mathcal{Y}_i \neq \emptyset$, for otherwise $\mathit{modes}_i = \emptyset$, which is a contradiction, or (alter) would be applicable to derive $\mathsf{tail}^k(\psi_i, \emptyset, \mathsf{N})$ from $\mathsf{tail}^k(\psi_i, \emptyset, A',)$ for some $A' \in \mathit{modes}_i$. But if $\mathcal{Y}_i \neq \emptyset$, then also $\mathcal{Y} \neq \emptyset$, whence there is no possible premise such that rule (alter) derives $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{N})$. But since $\mathsf{N} \notin \mathit{modes}_i$, rule $(\vee)$ is also not available, contradicting that $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{N})$ is derivable. Hence, $\mathsf{N} \in \mathit{modes}_i$ for $i \in \{1,2\}$ and rule $(\vee)$ is applicable with premises $\mathsf{tail}^k(\psi_i, \mathcal{Y}_i, \mathsf{N})$ for $i \in \{1,2\}$ and derives $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{N})$. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, \mathsf{N}, \mathsf{N})$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{Y}_i, \mathsf{N}, \varepsilon)$ for $i \in \{1,2\}$.

If $A = \mathsf{U}$, there are two cases. If $\mathcal{Y} = \emptyset$, then $\mathsf{N} \in \mathit{modes}$. Use rule (alter) to derive $\mathsf{tail}^k(\psi, \emptyset, \mathsf{U})$ from $\mathsf{tail}^k(\psi, \emptyset, \mathsf{N})$ and refer to the previous case. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, \mathsf{U}, \mathsf{N})$ and and continue with $\mathsf{info}(\psi_i) = (\mathcal{Y}_i, \mathsf{N}, \varepsilon)$ for $i \in \{1,2\}$. If $\mathcal{Y} \neq \emptyset$, then rule $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{N})$ can not be the conclusion of rule (alter), whence the only rule with this conclusion must be rule $(\vee_{\mathsf{U}})$. It follows that there is i such that $\mathcal{Y}_i = \emptyset$, and, moreover, $\mathsf{N} \in \mathit{modes}_1$ and $\mathsf{N} \in \mathit{modes}_2$, for otherwise $\mathsf{tail}^k(\psi, \mathcal{Y}, \mathsf{U})$ would not be derivable. Update $\mathsf{info}(\psi)$ to $(\mathcal{Y}, \mathsf{U}, \mathsf{U})$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{Y}_i, \mathsf{U}, \varepsilon)$ for $i \in \{1,2\}$.

Proceeding like this yields, for each subformula ψ in the formula tree of φ, a triple $\mathsf{info}(\psi) = (\mathcal{Y}, A, A')$ such that A is the relevant mode when connecting ψ with its predecessor, if it exists, and A' is the relevant mode when connecting ψ with its successors, if they exist. In particular, at least one rule is applicable with the selected modes in either direction.

Remark 5.2.3. As in the case of Algorithm 1, the above definition of $\mathsf{info}()$ can be adapted to strictly tail-recursive formulas by removing the first subcase from the definition in the case of fixpoint formulas.

Remark 5.2.4. The double pass algorithm to determine the exact derivation for an order-k tail recursive formula φ might seem over-engineered at first, but it is necessary to obtain an algorithm that generates the derivation in time linear in the size of the syntax tree of φ. A single-pass linear time algorithm would have to cope with the following problems: A bottom-up algorithm will not know which mode to assign to a fixpoint variable node, and a top-down algorithm cannot distinguish which mode to use given a boolean connective.

Consider the formula $\mu(X : \bullet).\,(X \vee X) \wedge P$. A bottom-up approach would not know which mode to assign to the subformulas X, and a top down approach would not know whether the conjunction should be assigned mode U, or whether advantage should be taken of the fact that the right conjunct is fixpoint free whence mode N is also possible. Our approach solves this by first running Algorithm 1. It generates all possible partial derivations bottom-up, even those that cannot be continued to a derivation for the full formula. Then it uses a top-down local approach to extract the exact derivation steps at each subformula, aided by information already collected by Algorithm 1 on which rules modes for the involved subformulas can actually be completed to a successful derivation.

5.2.2 Model-Checking Tail-Recursive Formulas

We construct a bounded-alternation k-EXPSPACE algorithm in order to model-check order-$(k+1)$ HFL-formulas that are order-$(k+1)$ tail-recursive. The recursion of least and greatest fixpoints is handled by a counter: Upon reaching a fixpoint definition, a counter is added to the fixpoint variable in question, indicating how many times it can be unfolded. Every time the variable is reached, the algorithm will continue with the defining formula of the fixpoint, but decrease the counter by one. Once the counter reaches zero, the algorithm terminates with the default value of true or false, depending on the polarity of the fixpoint. In order to connect this procedure to the semantics of HFL, consider the following definition:

Definition 5.2.5. Let $\mathcal{T}$ be an LTS, let φ be an order-k tail-recursive formula with fixpoint variables in $\mathcal{X}$. For $X \in \mathcal{X}$, let $\sigma_X.\, \mathsf{fp}_\varphi(X)$ be the defining formula for X.

Given an interpretation η and a mapping $\mathsf{count}\colon \mathcal{X} \mapsto \mathbb{N}$, define η^{count} as as

$$\begin{aligned}\eta^{\mathsf{count}}(x) &= \eta(x)\\ \eta^{\mathsf{count}}(X) &= [\![X^{\mathsf{count}(X)}]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} \text{ if } X \in \mathcal{X},\end{aligned}$$

where

$$\begin{aligned}
[\![X^{\mathsf{count}}]\!]_{\mathcal{T}}^{\eta} &= [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \mathtt{ff}]\!]_{\mathcal{T}}^{\eta} && \text{if } \mathsf{count}(X) = 0 \text{ and } \sigma_X = \mu\\
[\![X^{\mathsf{count}}]\!]_{\mathcal{T}}^{\eta} &= [\![\lambda x_1^X. \dots \lambda x_{k_X}^X.\, \mathtt{tt}]\!]_{\mathcal{T}}^{\eta} && \text{if } \mathsf{count}(X) = 0 \text{ and } \sigma_X = \nu\\
[\![X^{\mathsf{count}}]\!]_{\mathcal{T}}^{\eta} &= [\![\mathsf{fp}_\psi(X)]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}[X \mapsto n]}} && \text{if } \mathsf{count}(X) = n + 1.
\end{aligned}$$

Note that even though the definition appears to be circular, η^{count} is well-defined since $[\![X^{\mathsf{count}}]\!]_{\mathcal{T}}$ is either defined directly or in reference to $\eta^{\mathsf{count}[X \mapsto n]}$ where $n + 1 = \mathsf{count}(X)$.

For $\mathsf{count}\colon \mathcal{X} \to \mathbb{N}$ and $\mathsf{count}'\colon \mathcal{X} \to \mathbb{N}$ we define $\mathsf{count}' < \mathsf{count}$ as follows: $\mathsf{count}' < \mathsf{count}$ if there is a variable X such that $\mathsf{count}'(X) < \mathsf{count}(X)$ and $\mathsf{count}'(Y) = \mathsf{count}(Y)$ for all Y with $Y \succ X$. Moreover, if $\mathsf{count}(X) \neq 0$, we define $\mathsf{count}[X{-}{-}]$ as

$$\mathsf{count}[X{-}{-}](Y) = \begin{cases} \mathsf{count}(Y) \text{ if } Y \neq X\\ \mathsf{count}(X) - 1 \text{ if } Y = X.\end{cases}$$

Lemma 5.2.6. *Let $\mathcal{T}$ be a finite LTS. Let φ be an order-k tail-recursive formula and let $\psi = \sigma(X : \tau).\, \psi'$ be a subformula of φ. Then $[\![\sigma(X : \tau).\, \psi']\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} = [\![\psi']\!]_{\mathcal{T}}^{\eta^{\mathsf{count}[X \mapsto ht([\![\tau]\!]_{\mathcal{T}})]}}$.*

Proof. This is a direct consequence of the Kleene Fixpoint Theorem (Theorem 2.1.7).

We write $\emptyset$ for a mapping that maps each $X \in \mathcal{X}$ to $\mathsf{ht}([\![\tau]\!]_{\mathcal{T}})$. $\square$

Remark 5.2.7. Note that the above definitions around count are similar to those made around μ-signatures (see Section 3.2). In fact, the underlying principle of tracking approximations for all fixpoint variables in a formula is the same. However, in this case, we want to make the graph of the model-checking problem finite, as opposed to the setting of the model-checking game, where we explicitly wanted to generate an infinite play if $\mathcal{V}$ wins. Another difference is that, due to the syntactic restrictions of tail recursion, formulas with free fixpoint variables can never be bound to lambda variables and, hence, never must be substituted. It follows that one tuple of counters as in count is sufficient to correctly define semantics for a formula with free fixpoint variables, as opposed to the general case in Section 3.2, where each occurrence of each variable needed to be annotated with its own signature.

Consider Algorithm 2 and procedure VerTR in it. We claim that this procedure is a valid model-checking procedure for order-k tail-recursive formulas. Note that, apart from a formula in $\mathrm{HFL}^k_{\mathsf{tail}}$, it also needs as input a pointed LTS $\mathcal{T}, v_I$. Since both the order k and the LTS $\mathcal{T}$ are fixed during a run of the procedure, we do not display them explicitly to avoid notational bloat. Note that procedure VerTR is presented as an alternating algorithm using nondeterminism in mode N and co-nondeterminism in mode U. We mark nondeterministic, respectively co-nondeterministic transitions as "nondeterministically guess", respectively "universally choose". We argue in the proof of Theorem 5.2.10 why procedure VerTR has bounded alternation and how this can be used to obtain a deterministic algorithm in the desired complexity class.

In order to give the correctness proof for this claim, we need a measure of the degree to which a formula contains constructs that prohibit a simple run in the relevant mode, e.g., a straightforward nondeterministic model-checking procedure. Examples of such constructs are conjunctions while in mode N and, dually, disjunctions while in mode U, negations, applications etc. This measures how many nested non-tail-recursive calls are necessary during the algorithm.

Definition 5.2.8. Let φ be order-k tail-recursive. The *recursion depth* $rd(\psi)$ of a subformula ψ of φ is defined as $rd(\psi) = 0$ if $\mathsf{info}(\psi) = (\emptyset, \_, \mathsf{F})$ and otherwise as

- $rd(\psi) = 0$ if $\psi = P$ or $\psi = x$ or $\psi = X$,
- $rd(\psi) = rd(\psi')$ if $\psi = \langle a\rangle\psi'$ and $\mathsf{info}(\psi) = (\_, \_, \mathsf{N})$ or if $\psi = [a]\psi'$ and $\mathsf{info}(\psi) = (\_, \_, \mathsf{U})$ or if $\psi = \sigma X.\, \psi'$ or if $\psi = \lambda x.\psi'$,
- $rd(\psi) = 1 + rd(\psi')$ if $\psi = \langle a\rangle\psi'$ and $\mathsf{info}(\psi) = (\_, \_, \mathsf{U})$ or if $\psi = [a]\psi'$ and $\mathsf{info}(\psi) = (\_, \_, \mathsf{N})$ or if $\psi = \neg\psi'$,
- $rd(\psi) = \max\{r_1, r_2\}$ if $\psi = \psi_1 \vee \psi_2$ or $\psi = \psi_1 \wedge \psi_2$ where
$$r_i = \begin{cases} rd(\psi_i) \text{ if } \mathsf{info}(\psi_i) = (\mathcal{Y}, \_, \_) \text{ and } \mathcal{Y} \neq \emptyset \\ 1 + rd(\psi_i) \text{ if } \mathsf{info}(\psi_i) = (\emptyset, \_, \_) \end{cases}$$
for $i \in \{1, 2\}$,
- $rd(\psi) = \max\{rd(\psi_1), 1 + rd(\psi_2)\}$ if $\psi = \psi_1\, \psi_2$.

We are now ready to give the correctness proof for Algorithm 2.

Lemma 5.2.9. *Let $\mathcal{T}, v_I$ be a finite LTS and let φ be a closed* HFL *formula of ground type that is order-$(k+1)$ tail-recursive. Let $\mathsf{info}(\varphi) = (\emptyset, A, \_)$. Then $\mathsf{VerTR}(A, v_I, \varphi, \varepsilon, \emptyset, \emptyset)$ terminates and returns* true *if and only if $\mathcal{T}, v_I \models \varphi$.*

Proof. We are now going to prove by induction the following statement: If ψ is a subformula of φ of type $\tau_1 \to \cdots \to \tau_n \to \bullet$ and $\mathsf{info}(\psi) = (\_, A, A')$ and $f_1, \ldots, f_n$ with $f_i \in [\![\tau_i]\!]_{\mathcal{T}}$ for $1 \leq i \leq n$ and $A'' \in \{A, A'\}$ then

$$v \in (\cdots([\![\psi]\!]^{\eta^{\mathsf{count}}}_{\mathcal{T}}\ f_1) \cdots f_n) \text{ if and only if } \mathsf{VerTR}(A'', v, \psi, (f_1, \ldots, f_n), \eta, \mathsf{count}) \text{ returns } \mathsf{true}$$

and terminates.

Algorithm 2 Efficient model-checking for order-$(k+1)$ tail-recursive HFL formulas.

```
1: procedure $\mathrm{MCtr}(A, v, \psi, (f_1, \ldots, f_k), \eta, \mathsf{count})$
2:         ▷ Inputs $\mathcal{T} = (S, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ and $k = ord(\psi)$ are tacitly available as global variables
3:     $(\mathcal{Y}, A'', A') \leftarrow \mathsf{info}(\psi)$
4:     if $A \neq A'$ then return $MCtr(A', v, \psi, (f_1, \ldots, f_k), \eta, \emptyset)$
5:     if $A = \mathsf{F}$ then
6:         $f \leftarrow [\![\psi]\!]^{\eta}_{\mathcal{T}}$                ▷ use a conventional model checker
7:         if $v \in ((f\, f_1) \cdots f_k)$ then return true
8:         else return false
9:     switch $\psi$ do
10:        case $\psi = P$
11:            if $\mathcal{T}, v \models P$ then return true
12:            else return false
13:        case $\psi = x$
14:            $f \leftarrow \eta(x)$
15:            if $v \in ((f\, f_1), \cdots f_k)$ then return true
16:            else return false
17:        case $\psi = \psi_1 \vee \psi_2$
18:            if $A = \mathsf{N}$ then
19:                nondeterministically guess $i \in \{1, 2\}$
20:                return $MCtr(A, v, \psi_i, (f_1, \ldots, f_k), \eta, \mathsf{count})$
21:            else if $A = \mathsf{U}$ then
22:                $(\mathcal{Y}_i, \_, A_i) \leftarrow \mathsf{info}(\psi_i), i \in \{1, 2\}$
23:                choose $i \in \{1, 2\}$ s.t. $\mathcal{Y}_i = \emptyset$
24:                $b \leftarrow MCtr(A_i, v, \psi_i, (f_1, \ldots, f_k), \eta, \emptyset)$
25:                if $b = \mathsf{true}$ then return true
26:                else return $MCtr(A, v, \psi_{1-i}, (f_1, \ldots, f_k), \eta, \mathsf{count})$
27:        case $\psi = \psi_1 \wedge \psi_2$
28:            if $A = \mathsf{U}$ then
29:                universally choose $i \in \{1, 2\}$
30:                return $MCtr(A, v, \psi_i, (f_1, \ldots, f_k), \eta, \mathsf{count})$
31:            else if $A = \mathsf{N}$ then
32:                $(\mathcal{Y}_i, \_, A_i) \leftarrow \mathsf{info}(\psi_i), i \in \{1, 2\}$
33:                guess $i \in \{1, 2\}$ s.t. $\mathcal{Y}_i = \emptyset$
34:                $b \leftarrow MCtr(A_i, v, \psi_i, (f_1, \ldots, f_k), \eta, \emptyset)$
35:                if $b = \mathsf{false}$ then return false
36:                else return $MCtr(A, v, \psi_{1-i}, (f_1, \ldots, f_k), \eta, \mathsf{count})$
37:        case $\psi = \neg\psi'$
38:            $b \leftarrow MCtr(A, v, \psi', (f_1, \ldots, f_k), \eta, \emptyset)$
39:            if $b = \mathsf{true}$ then return false
40:            else return true
```

```
case $\psi = \langle a \rangle \psi'$
    if $A = \mathsf{N}$ then
        nondeterministically guess $w$ with $v \xrightarrow{a} w$
        return $MCtr(A, w, \psi', (f_1, \dots, f_k), \eta, \mathsf{count})$
    else if $A = \mathsf{U}$ then
        for $w$ with $v \xrightarrow{a} w$ do
            $b \leftarrow MCtr(A, w, \psi', (f_1, \dots, f_k), \eta, \emptyset)$
            if $b = \mathsf{true}$ then return $\mathsf{true}$
        return $\mathsf{false}$
case $\psi = [a] \psi'$
    if $A = \mathsf{U}$ then
        universally choose $w$ with $v \xrightarrow{a} w$
        return $MCtr(A, w, \psi', (f_1, \dots, f_k), \eta, \mathsf{count})$
    else if $A = \mathsf{N}$ then
        for $w$ with $v \xrightarrow{a} w$ do
            $b \leftarrow MCtr(A, w, \psi', (f_1, \dots, f_k), \eta, \emptyset)$
            if $b = \mathsf{false}$ then return $\mathsf{false}$
        return $\mathsf{true}$
case $\psi = \psi_1\, \psi_2$
    $(\mathcal{Y}_2, \_, A_2') \leftarrow \mathsf{info}(\psi_2)$
    if $A_2' = \mathsf{F}$ then
        $f \leftarrow [\![\psi_2]\!]_{\mathcal{T}}^{\eta}$                    ▷ use a conventional model checker
        return $MCtr(A, v, \psi_1, (f, f_1, \dots, f_k), \eta, \mathsf{count})$
    else
        $(\tau_1 \to \cdots \to \tau_n \to \bullet) \leftarrow \tau(\psi_2)$
        $f \leftarrow \emptyset$
        for $g_1, \dots, g_n \in [\![\tau_1]\!]_{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]_{\mathcal{T}}$ do
            $T \leftarrow \emptyset$
            for $w \in S$ do
                $b \leftarrow MCtr(A_2, w, \psi_2, (g_1, \dots, g_n), \eta, \emptyset)$
                if $b = \mathsf{true}$ then $T \leftarrow T \cup \{w\}$
            $f \leftarrow f[(g_1, \dots, g_n) \mapsto T]$
        return $MCtr(A, v, \psi_1, (f, f_1, \dots, f_k), \eta, \mathsf{count})$
case $\psi = \lambda x.\, \psi'$
    return $MCtr(A, v, \psi', (f_2, \dots, f_k), \eta[x \mapsto f_1], \mathsf{count})$
case $\psi = \sigma(X : \tau).\, \psi'$
    return $MCtr(A, v, \psi', (f_1, \dots, f_k), \eta, \mathsf{count}[X \mapsto \mathsf{ht}([\![\tau]\!]_{\mathcal{T}})])$
case $\psi = X$
    if $\mathsf{count}(X) \neq 0$ then
        return $MCtr(A, v, (\mathsf{fp}_{\varphi}(X), (f_1, \dots, f_k), \eta, \mathsf{count}[X\text{--}])$
    else if $\sigma_X = \mu$ then return $\mathsf{false}$
    else return $\mathsf{true}$
```

The statement of the lemma then follows with $\psi = \varphi$, $v = v_I$, $n = 0$, $\eta = \emptyset$ and $\mathsf{count} = \emptyset$.

The induction has four induction parameters: rd, ψ, count and A. We then show that, if $\mathsf{VerTR}(A', \_, \psi', \_, \_, \mathsf{count}')$ is called tail-recursively during evaluation of $\mathsf{VerTR}(A, \_, \psi, \_, \_, \mathsf{count})$, i.e., in the form **return** $\mathsf{VerTR}(\dots)$, then either

- $\mathsf{count} = \mathsf{count}'$, $\psi = \psi'$ and $\mathsf{info}(\psi) = (\_, A, A')$ or
- $\mathsf{count} = \mathsf{count}'$ and ψ' is a proper subformula of ψ or
- $\mathsf{count}' < \mathsf{count}$.

Moreover, no such call during the algorithm will increase recursion depth, and if such a call of $\mathsf{VerTR}(A', \_, \psi', \_, \_, \mathsf{count}')$ is not tail recursive in an algorithmic sense, i.e., if it is of the form $b \leftarrow \mathsf{VerTR}(\dots)$ then $rd(\psi') < rd(\psi)$.

Let $\mathsf{VerTR}(A, v, \psi, (f_1, \dots, f_k), \eta, \mathsf{count})$ be a call of VerTR. Furthermore, let $(\mathcal{Y}, A', A'') = \mathsf{info}(\psi)$. If $A \neq A''$ then the algorithm returns the value of

$$\mathsf{VerTR}(A'', v, \psi, (f_1, \dots, f_n), \eta, \mathsf{count})$$

which, by the induction hypothesis is true if and only if $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$, which is also the claim of the lemma for $\mathsf{VerTR}(A, v, \psi, (f_1, \dots, f_k), \eta, \mathsf{count})$. Now assume that $A = A''$. If $A = \mathsf{F}$ then $\mathcal{Y} = \emptyset$ and VerTR calls a conventional model checker to compute $[\![\psi]\!]_{\mathcal{T}}^{\eta}$, which works correctly by assumption. The claim of the lemma then follows.

If $A \neq \mathsf{F}$, the argument depends on the form of ψ:

- If ψ is of the form P, or x, then the claim of the lemma is immediate.
- If $\psi = \psi_1 \vee \psi_2$, there are two cases: If $A = \mathsf{N}$, then VerTR guesses $i \in \{1, 2\}$ and returns the value of $\mathsf{VerTR}(A, v, \psi_i, \varepsilon, \eta, \mathsf{count})$. Hence, VerTR returns true if and only if there is i such that $\mathsf{VerTR}(A, v, \psi_i, \varepsilon, \eta, \mathsf{count})$ returns true, which, by the induction hypothesis is the case if and only if $v \in [\![\psi_i]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$. By the definition of HFL semantics, $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$ if and only if $v \in [\![\psi_i]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$ for at least one $i \in \{1, 2\}$ whence $\mathsf{VerTR}(\mathsf{N}, v, \psi, \varepsilon, \eta, \mathsf{count})$ returns true if and only if $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$.

  If $A = \mathsf{U}$, then VerTR universally chooses $i \in \{1, 2\}$ such that $\mathsf{info}(\psi_i) = (\mathcal{Y}_i, \_, A_i)$ and $\mathcal{Y}_i = \emptyset$ and calculates $b = \mathsf{VerTR}(A_i, v, \psi_i, \varepsilon, \eta, \emptyset)$. By the induction hypothesis, $b = \mathsf{true}$ if and only if $v \in [\![\psi_i]\!]_{\mathcal{T}}^{\eta^{\emptyset}}$ which also entails $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$. So in case $b = \mathsf{true}$, the algorithm works as claimed in the lemma. Note that also, because $\mathcal{Y}_i = \emptyset$, we have that $rd(\psi_i) < rd(\psi)$ so the condition on non-tail recursive calls is satisfied. In case $b = \mathsf{false}$, the algorithm returns the value of $\mathsf{VerTR}(A, v, \psi_{1-i}, \varepsilon, \eta, \mathsf{count})$. By the induction hypothesis, this return value is true if and only if $v \in [\![\psi_{i-1}]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$ and, by the definition of HFL semantics, this is the case if and only if $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$, which settles the claim of the lemma.
- If $\psi = \psi_1 \wedge \psi_2$, the argument is analogous to the case where $\psi = \psi_1 \vee \psi_2$.

- If $\psi = \langle a \rangle \psi'$ there are two cases. If $A = \mathsf{N}$, then VerTR guesses w with $v \xrightarrow{a} w$ and returns the value of $\mathsf{VerTR}(A, w, \psi', \varepsilon, \eta, \mathsf{count})$. By the induction hypothesis, that value is true if and only if $w \in [\![\psi']\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$ which, by the definition of HFL semantics, entails that $\mathsf{VerTR}(\mathsf{N}, v, \psi, \varepsilon, \eta, \mathsf{count})$ returns true if and only if $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$.

  If $A = \mathsf{U}$, the algorithm iterates through all w with $v \xrightarrow{a} w$ and returns true if and only if $\mathsf{VerTR}(A, w, \psi', \varepsilon, \eta, \emptyset)$ returns true. If this is the case for some such w, then by the induction hypothesis, $w \in [\![\psi']\!]_{\mathcal{T}}^{\eta^{\emptyset}}$ and, by the definition of HFL semantics, also $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$. Hence, $\mathsf{VerTR}(\mathsf{U}, v, \psi, \varepsilon, \eta, \mathsf{count})$ returns true if and only if $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$. Note that necessarily $\mathsf{info}(\psi') = (\emptyset, \_, \_)$ and, hence, that $rd(\psi') < rd(\psi)$, so the condition on calls that are not tail recursive is satisfied.

- If $\psi = [a]\psi'$, again the argument is analogous to the case where $\psi = \langle a \rangle \psi'$.

- If $\psi = \neg\psi'$ then the algorithm computes $b = \mathsf{VerTR}(A, v, \psi', \varepsilon, \eta, \emptyset)$. By the induction hypothesis, $b = \mathsf{true}$ if and only if $v \in [\![\psi']\!]_{\mathcal{T}}^{\eta^{\emptyset}}$. Hence, by the definition of HFL semantics, $\mathsf{VerTR}(A, v, \psi, \varepsilon, \mathsf{count})$ returns true if and only if $v \in [\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}$. Note that necessarily $\mathsf{info}(\psi') = (\emptyset, \_, \_)$ and, hence $rd(\psi') < rd(\psi)$ whence the condition on calls that are not tail recursive is satisfied.

- If $\psi = \psi_1\, \psi_2$, let $(\emptyset, \_, A_2) = \mathsf{info}(\psi_2)$. If $A_2 = \mathsf{F}$, then the algorithm calls a conventional model-checker to determine $f = [\![\psi_2]\!]_{\mathcal{T}}^{\eta^{\emptyset}}$. The full semantics of a function type are computed by calling the model-checker consecutively on each possible argument in order to build the full function table representing the semantics of the function type.

  If $A_2 \neq \mathsf{F}$, let $\tau_1 \to \cdots \to \tau_n \to \bullet$ be the type of ψ_2. Let $(\emptyset, \_, A_2) = \mathsf{info}(\psi_2)$. The algorithm then computes, for each $g_1, \ldots, g_n$ in $[\![\tau_1]\!]_{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]_{\mathcal{T}}$, and each vertex w, whether $\mathsf{VerTR}(A_2, w, \psi_2, (g_1, \ldots, g_n), \eta, \emptyset)$ returns true. Since necessarily $rd(\psi_2) < rd(\psi)$, the condition on calls that are not tail recursive is satisfied and, by the induction hypothesis this call returns true if and only if $w \in [\![\psi_2]\!]_{\mathcal{T}}^{\eta}$. Hence, $T = \{w \mid \mathsf{VerTR}(A_2, w, \psi_2, (g_1, \ldots, g_k), \eta, \emptyset) = \mathsf{true}\}$ is equal to $(\cdots([\![\psi_2]\!]_{\mathcal{T}}^{\eta}\, g_1) \cdots\, g_n)$, and updating f to map $g_1, \ldots, g_n$ to T yields that $(\cdots(f\, g_1) \cdots\, g_n) = (\cdots([\![\psi_2]\!]_{\mathcal{T}}^{\eta}\, g_1) \cdots\, g_n)$. By repeating this process for all $g_1, \ldots, g_n \in [\![\tau_1]\!]_{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]_{\mathcal{T}}$, we obtain that $f = [\![\psi_2]\!]_{\mathcal{T}}^{\eta}$.

  In both cases, the algorithm then returns the value of

$$\mathsf{VerTR}(A, v, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{count}),$$

  which by the induction hypothesis is true if and only if

$$v \in (\cdots(([\![\psi_1]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}\, f)\, f_1) \cdots\, f_k),$$

  and the latter holds if and only if $v \in (\cdots([\![\psi]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}\, f_1) \cdots)\, f_k)$. The claim of the lemma follows.

- If $\psi = \lambda x.\psi'$, then the claim of the lemma is immediate.

- If $\psi = \sigma(X : \tau).\, \psi'$, then the algorithm returns the value of

$$\mathsf{VerTR}(A, v, \psi', (f_1, \ldots, f_k), \eta, \mathsf{count}[X \mapsto \mathsf{ht}([\![\tau]\!]_{\mathcal{T}})]).$$

  By the induction hypothesis, this value is true if and only if

$$v \in (\cdots([\![\psi']\!]_{\mathcal{T}}^{\eta^{\mathsf{count}[X \mapsto \mathsf{ht}([\![\tau]\!]_{\mathcal{T}})]}} f_1) \cdots f_k),$$

  which, by Lemma 5.2.6 is equivalent to $v \in (\cdots([\![\sigma(X : \tau).\, \psi']\!]_{\mathcal{T}}^{\mathsf{count}} f_1) \cdots f_k)$.

- If $\psi = X$ and $\mathsf{count}(X) = 0$ then note that $[\![X]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} = X^{\mathsf{count}(X)}$. If $\sigma_X = \mu$ then $X^{\mathsf{count}(X)} = \mathtt{ff}^\tau$ where τ is the type of X. Hence, we have that $v \notin (\cdots([\![\mathtt{ff}^\tau]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} f_1) \cdots f_k)$, and the algorithm correctly returns false. If $\sigma_X = \nu$ then $[\![X]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} = \mathtt{tt}^\tau$, where τ is the type of X. Hence, we have that $v \in (\cdots([\![\mathtt{tt}^\tau]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} f_1) \cdots f_k)$ and the algorithm correctly returns true.

  On the other hand, if $\mathsf{count}(X) \neq 0$, then the algorithm returns the value of $\mathsf{VerTR}(A, v, \mathsf{fp}_\varphi(X), \eta, \mathsf{count}[X\text{--}])$. Note that

$$[\![X]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} = \eta^{\mathsf{count}}(X) = [\![X^{\mathsf{count}(X)}]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} = [\![\mathsf{fp}_\varphi(X)[X^{\mathsf{count}[X\text{--}]}/X]]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}}.$$

  However, since X does not appear freely in $\mathsf{fp}_\varphi(X)[X^{\mathsf{count}[X\text{--}]}/X]$, this is the same as $[\![\mathsf{fp}_\varphi(X)[X^{\mathsf{count}[X\text{--}]}/X]]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}[X\text{--}]}} = [\![\mathsf{fp}_\varphi(X)]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}[X\text{--}]}}$. It follows that $v \in (\cdots([\![X]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}}} f_1) \cdots f_k)$ if and only if $v \in (\cdots([\![\mathsf{fp}_\varphi(X)]\!]_{\mathcal{T}}^{\eta^{\mathsf{count}[X\text{--}]}} f_1) \cdots f_k)$. Since $\mathsf{count}[X\text{--}] < \mathsf{count}$, by the induction hypothesis this is true if and only if

$$\mathsf{VerTR}(A, v, \mathsf{fp}_\psi(X), (f_1, \cdots, f_k), \eta, \mathsf{count}[X\text{--}])$$

  returns true. It follows that $\mathsf{count}(X) \neq 0$. From that we then obtain that the call $\mathsf{VerTR}(A, v, \psi, (f_1, \ldots, f_k), \eta, \mathsf{count})$ returns true if and only if also $\mathsf{VerTR}(A, v, \mathsf{fp}_\varphi(X), (f_1, \ldots, f_k), \eta, \mathsf{count}[X\text{--}])$ returns true.

□

Theorem 5.2.10. *The model-checking problem for order-$(k+1)$ tail-recursive* HFL *is in k-EXPSPACE.*

Note that Algorithm 2 is an alternating algorithm. However, the number of alternations is bounded. We explain at the end of the proof how bounded alternation can be used to obtain a deterministic version of Algorithm 2.

Proof. Calls to a conventional model-checker are made for formulas that do not contain order-$(k+1)$ fixpoint definitions. While such formulas are not necessarily of order k or lower, their only order-$(k+1)$ elements are lambda abstractions. Using a model-checking strategy that, given an application, always computes the semantics of the operand before computing the semantics of the operator, the subformula in question can be model-checked in k-EXPTIME. Hence, we can safely assume that any such calls to a conventional model-checker conclude well within the desired complexity bounds.

The information that needs to be stored for the evaluation of an instance of $\mathsf{VerTR}(A, v, \psi, (f_1, \ldots, f_n), \eta, \mathsf{count})$ takes k-fold exponential space: references to a mode, a vertex and a subformula take linear space, each of the function tables

$f_1, \dots, f_n$ appears in operand position and, hence, is a function of order at most k, which takes k-fold exponential space. An environment is just a partial map from $\mathcal{F}$ to more function tables, also of order at most k. Finally, count stores at most $|\mathcal{X}|$ many numbers whose values are bounded by an $(k+1)$-fold exponential. Hence, they can be represented as k-fold exponentially long bit strings.

During evaluation, VerTR operates in a tail-recursive fashion for most operators, which means that no stack has to be maintained and the space needed is restricted to what is described in the previous paragraph. A calling context (which is just an instance of VerTR as described above, with an added logarithmically sized counter in case of $[a]\psi$ and $\langle a\rangle\psi$) has to be preserved only at steps of the form $b \leftarrow \mathsf{VerTR}(\dots)$, in which case the call goes to a subformula with strictly smaller recursion depth. Since the recursion depth of an $\mathrm{HFL}^k_{\mathsf{tail}}$-formula is linear in the size of the formula, only linearly many such calling contexts have to be stored at any given point during the evaluation, which does not exceed nondeterministic k-fold exponential space. It follows that VerTR runs in k-fold exponential space.

Regarding alternation, note that alternation occurs on two places: It occurs if $\mathsf{VerTR}(A, v, \psi, (f_1, \dots, f_k), \eta, \mathsf{count})$ is called such that $\mathsf{info}(\psi) = (\emptyset, A, A')$ with $A' \neq A$, and potentially for tail-recursive calls. Since the first kind of alternation necessarily occurs in subformulas that are fixpoint variable closed, and the latter occurs only for fixpoint closed subformulas with strictly decreasing recursion depth, the maximum nesting depth of alternation is bounded by the size of the input formula. Hence, we can apply Theorem 4.2 from [24] (a generalization of Savitch's Theorem attributed to Borodin), which states that an algorithm with that needs at most k-fold exponential space and has at most k-fold exponentially many alternations can be simulated in deterministic k-EXPSPACE. Hence, VerTR can be simulated in deterministic k-EXPSPACE. $\square$

Since order-$(k+1)$ strictly tail-recursive HFL is a fragment of order-$(k+1)$ tail-recursive HFL, the previous theorem also yields a similar upper bound for the former.

Corollary 5.2.11. *The model-checking problem for strictly tail-recursive* HFL *of order-*$(k+1)$ *is in* k-EXPSPACE.

Remark 5.2.12. Note that for the invocation of Theorem 4.2 from [24], we defined the number of alternations as the maximum nesting depth of alternation during a recursive call, which is bounded by the recursion depth of the input formula. This is the correct measure for a recursive procedure. A naive application of the definition given in [24] makes the result fail: If one considers the Turing Machine equivalent of a run of procedure VerTR then non-tail-recursive calls, for example at an application, behave as follows: The machine computes the run up to the non-tail-recursive call, then computes the call itself, and then continues with the main procedure. Hence, alternations in all three parts must be considered additively towards the total number of alternations in the procedure. In particular for applications this produces way too many alternations so that Theorem 4.2 cannot be applied. Another example would be the formula

$$\big(\mu(X^{\cdot} \colon \bullet \to \bullet)\, \lambda(x \colon \bullet).\ (\langle a\rangle X\, x \vee \langle b\rangle X\, x) \wedge x\big)\ P$$

which has a non-tail-recursive call at the conjunction. During a run that makes $\mathsf{count}(X)$ reach 0, the naive approach will witness $\mathsf{ht}(\llbracket \bullet \to \bullet \rrbracket_{\mathcal{T}})$ many alternations due to the non-tail-recursive call at the conjunction, where $\mathcal{T}$ is the LTS in question. In general, this is an exponentially large number, even though the algorithm is intended to run in PSPACE. However, since Theorem 4.2 from [24] is a generalization of Savitch's Theorem [80], the proof proceeds in a similar manner. More precisely, it proceeds by enumerating all possible configurations where the algorithm terminates successfully, and then checking whether one of them can be reached from the initial configuration in a given number of steps. The latter is done by enumerating all possible intermediate configurations and then verifying that one of them can be reached from the initial configuration in half as many steps, while simultaneously a given final configuration can also be reached in half as many steps. Procedure VerTR lends itself very naturally to this approach. Given a non-tail-recursive call, the problem amounts to deciding whether a given configuration immediately after the non-tail-recursive call can be reached from the configuration before it, given the result of the call. This, in turn, reduces to another instance of the procedure outlined above. Hence, we get a chained invocation of Savitch's Theorem. In particular, only the problem of whether the configuration after the non-tail-recursive call can be reached from the preceding configuration depends on that chained invocation. Since the maximal depth of such chained instances is bounded by the alternation depth of the input formula, which is at most linear in the size of the formula, we get at most linearly many chained instances of Savitch's Theorem, which clearly does not exhaust the complexity limitations. We omit a formal proof since this approach appears to be the natural adaption of the result in [24] towards a mostly tail-recursive algorithm where the maximal nesting depth of non-tail-recursive calls is bounded by a small number.

Remark 5.2.13. The lowered complexity of the model-checking problem for tail-recursive HFL is in line with the observations of Emerson et al. [35], who exhibit a fragment of $\mathcal{L}_\mu$ with very similar restrictions with respect to boolean alternation, and show that the complexity of its model-checking problem is a small polynomial rather than the full complexity of the $\mathcal{L}_\mu$-model-checking problem, whose complexity is still to be determined tightly.

5.3 Lower Bounds for Model Checking

Model-Checking order-$(k+1)$ tail-recursive HFL, even in the strict variant, is complete for k-fold exponential space. The hardness result already holds for data complexity. In order to show the lower bound, for $k > 0$ we reduce a problem called the order-k corridor tiling problem to the model-checking problem for order-$(k+1)$ (strictly) tail-recursive HFL formulas. The former is known to be complete for k-fold exponential space. The case of $k = 0$ is done differently. We chose the corridor tiling problem since it can be encoded into HFL formulas with less coding than e.g., runs of deterministic space-bounded Turing Machines.

5.3.1 The Corridor Tiling Problem

Intuitively, the order-k corridor tiling problem (see e.g.. [88]) consists of a finite number of so-called tiles which pairwise either match or do not match horizontally or vertically, and asks to construct a finite sequence of rows of tiles each of a given width of k-fold exponential size in some given number n, such that adjacent tiles in a given row, respectively in adjacent rows match horizontally, respectively vertically. Such a tiling is successful if the last row begins with a designated final tile. Formally, a *tiling system* $\mathcal{K}$ is of the form $\mathcal{K} = (T, H, V, t_I, t_\Box, t_F)$ where T is a finite set of *tiles*, $H, V \subseteq T \times T$ are the *horizontal* and *vertical matching relations* and $t_I, t_\Box, t_F \in T$ are designated tiles called the *initial*, *boundary* and *final* tiles.

Given a tiling system $\mathcal{K}$ and a natural number n, the order-k corridor tiling problem is then to decide whether there is a *successful tiling* of width 2^n_k. A successful tiling is a finite sequence $(r_i)_{i \leq m}$ of *rows*, i.e., words in T^* of length 2^n_k each, such that

- the first row is $t_I t_\Box \cdots t_\Box$,
- the last row begins with t_F,
- if the ith row is $t_1, \dots, t_{2^n_k}$ then, for all $0 \leq j \leq 2^n_k - 2$, the pair (t_j, t_{j+1}) is in H,
- if the ith row is $t_1, \dots, t_{2^n_k}$ and the $(i+1)$st row is $t'_1, \dots, t'_{2^n_k}$, then, for all $0 \leq j \leq 2^n_k - 1$, the pair (t_j, t'_j) is in V.

We now have the following result:

Theorem 5.3.1. *Given a tiling system $\mathcal{K}$ with m tiles and $n \geq m$ presented in unary, it is a k-*EXPSPACE*-hard problem to determine whether $\mathcal{K}$ has a successful tiling of width 2^n_k for $k \geq 0$.*

This result appears in a survey of van Emde Boas [88] for $k = 0$, i.e., for PSPACE hardness. The idea is that each row encodes a configuration of a PSPACE Turing Machine given by the tiling system. A successful tiling then encodes a successful run. For a more thorough presentation of the reduction see [88] or [30].

Our goal is now to construct, for each $k > 1$, a strictly tail-recursive HFL formula $\varphi_{k+1,n}$ of order $k+1$ such that the model-checking problem for φ_{k+1} is k-EXPSPACE-hard. This formula will encode solutions to the order-k corridor tiling problem in the sense that for all pointed LTS $\mathcal{T}, v$ that encode such a tiling system, $\mathcal{T}, v \models \varphi_{k+1,n}$ if and only if the associated tiling problem has a successful tiling of width 2^n_k. We now show how an LTS encodes a tiling system. The formula $\varphi_{k+1,n}$ will be constructed later.

Fix a set of transitions $A = \{\mathsf{u}, \mathsf{d}, \mathsf{e}, \mathsf{h}, \mathsf{v}\}$ and a set of propositions $\{P_I, P_\Box, P_F\}$. Let $\mathcal{K} = (K, H, V, t_I, t_\Box, t_F)$ be a tiling system such that $|K| = m$. Without loss of generality, $K = \{t_0, \dots, t_{m-1}\}$ such that $t_0 = t_I$ and $t_{m-2} = t_\Box$ and $t_{m-1} = t_F$.

We say that $\mathcal{T}$ encodes $\mathcal{K}$ if $\mathcal{T} = (\{0, \dots, n-1\}, (\xrightarrow{a} \mid a \in A), \mathcal{L})$ where $n \geq m$ and $(\xrightarrow{a} \mid a \in A)$ is such that

- $i \xrightarrow{\mathsf{u}} j$ if and only if $j > i$,
- $i \xrightarrow{\mathsf{d}} j$ if and only if $j < i$,
- $i \xrightarrow{\mathsf{e}} j$ for all $0 \leq i, j \leq n-1$,
- $i \xrightarrow{\mathsf{h}} j$ if and only if $(t_i, t_j) \in H$,
- $i \xrightarrow{\mathsf{v}} j$ if and only if $(t_i, t_j) \in V$,

and

$$\mathcal{L}(i) = \begin{cases} \{P_I\} \text{ if } i = 0 \\ \{P_\Box\} \text{ if } i = m-2 \\ \{P_F\} \text{ if } i = m-1 \\ \emptyset \text{ otherwise.} \end{cases}$$

Intuitively, the vertices $0, \dots, m-1$ encode the tiles, the propositional labeling identifies the special tiles, and the transitions in h and v encode the horizontal and vertical matching relations. The transitions in u and d lead to vertices with a higher, respectively lower number while e simulates a global transition relation. The last three relations are used to encode numbers as described below. Hence, vertices in such an LTS play a dual role: they serve as encodings of the tiles in the tiling system, and will also be used to encode large numbers.

Clearly, for every $n \geq m$ there is an LTS with n vertices that encodes $\mathcal{K}$.

5.3.2 Jones's Encoding of Large Numbers for HFL

Fix a a tiling system $\mathcal{K}$ and some n greater than the number of tiles in $\mathcal{K}$. In order to encode a row in the order-k corridor tiling problem for some n, we need some additional machinery. Mathematically speaking, such a row, as a word of length 2^n_k, is a function from the set $\{0, \dots, 2^n_k - 1\}$ into the set of tiles of the respective tiling system. Such a representation requires the ability to encode large numbers. Since an LTS of size n that encodes $\mathcal{K}$ is ordered via the transition relations u and d, it is possible to exploit this order by interpreting the vertices of the LTS as bits. Recall that such an LTS has vertices $0, \dots, (n-1)$. A set T of vertices then represents the number $\Sigma_{i \in T} 2^i$, where the vertex 0 is considered the least significant bit. Obviously, in an LTS with n vertices this approach only makes it possible to encode numbers up to $2^n - 1$. We lift this approach by considering a function of order $k+1$ to encode a number if it returns the full set of vertices or the empty set of vertices on all inputs of order k that themselves encode a number. This idea is due to Jones [49].

For a formal definition, let τ_k be defined as $\tau_0 = \bullet$ and $\tau_{k+1} = \tau_k \to \bullet$. Given an LTS $\mathcal{T}$ of size n encoding $\mathcal{K}$, define $\top_\mathcal{T} = \{0, \dots, n-1\}$ and $\bot_\mathcal{T} = \emptyset$.

Definition 5.3.2. For some object $f \in [\![\tau_k]\!]_\mathcal{T}$, $\mathsf{jones}_k(f)$ is potentially defined to be a number in $0, \dots, 2^n_k - 1$ as follows:

- If $k = 0$ then $f \in [\![\bullet]\!]_\mathcal{T}$, $\mathsf{jones}_0(f)$ is always defined as $\mathsf{jones}_0(f) = \Sigma_{i \in f} 2^i$.
- If $k > 0$, then $\mathsf{jones}_k(f)$ is defined if and only if for all $i \in \{0, \dots, 2^n_{k-1} - 1\}$ and all $f_1, f_2 \in [\![\tau_k]\!]_\mathcal{T}$ with $\mathsf{jones}_{k-1}(f_1) = \mathsf{jones}_{k-1}(f_2) = i$ we have $f\, f_1 = f\, f_2$ and, moreover, the value of $f\, f_1$ is either $\top_\mathcal{T}$ or $\bot_\mathcal{T}$. In this case, let $B = \{i \in \{0, \dots, 2^n_k - 1 \mid f\, f' = \top_\mathcal{T}$ for some $f' \in [\![\tau_{k-1}]\!]_\mathcal{T}$ with $\mathsf{jones}_k(f') = i\}$ and $\mathsf{jones}_k(f) = \Sigma_{i \in B} 2^i$.

We write $\mathsf{jones}_k(f) = i$ to denote that $\mathsf{jones}_k(f)$ is defined and equals i.

The idea here is that a set T, i.e., an element of $[\![\tau_0]\!]_\mathcal{T}$ encodes a number in $\{0, \dots, 2^n_1 - 1\}$ as outlined above. This means that the bits of $\mathsf{jones}_0(T)$ are the individual vertices of the LTS, and a bit is set if the vertex is in T. Note that every set of vertices encodes a number. Following the scheme, a function in $[\![\tau_1]\!]_\mathcal{T}$ encodes a number in $0, \dots, 2^n_2 - 1$ if at every argument, it returns $\top_\mathcal{T}$ or $\bot_\mathcal{T}$. Hence, the bits of functions in $[\![\tau_1]\!]_\mathcal{T}$ are sets, i.e., objects in $[\![\tau_0]\!]_\mathcal{T}$. However, not every function in $[\![\tau_1]\!]_\mathcal{T}$ encodes a number, since there are functions that do not return $\top_\mathcal{T}$ or $\bot_\mathcal{T}$ at every argument. Starting at τ_2, there are several functions that encode the same number: Given a function f in $[\![\tau_2]\!]_\mathcal{T}$ such that $f\,f'$ is $\top_\mathcal{T}$ or $\bot_\mathcal{T}$ for all f' in $[\![\tau_1]\!]_\mathcal{T}$ that themselves encode a number, the value of f at arguments that do not encode a number themselves can be completely arbitrary. Hence, two such functions in $[\![\tau_2]\!]_\mathcal{T}$ that agree on their bits, i.e., the functions in $[\![\tau_1]\!]_\mathcal{T}$ that themselves encode a number, will encode the same number in $\{0, \dots, 2^n_3 - 1\}$ even if the do not agree on the other values. This motivates the stipulation that an object in $[\![\tau_k]\!]_\mathcal{T}$ for $k > 0$ encodes a number only if it returns the same value for all objects in $[\![\tau_{k-1}]\!]_\mathcal{T}$ that encode the same number.

Note that for each $k \geq 1$ and each $i \in \{0, \dots, 2^n_{k+1} - 1\}$, there is at least one $f \in [\![\tau_k]\!]_\mathcal{T}$ such that $\mathsf{jones}(f) = i$, for example the function f that sends all bits set in the binary representation of i to $\top_\mathcal{T}$ and everything else to $\bot_\mathcal{T}$. Formally, this is the function f defined via

$$f\,f' = \begin{cases} \bot_\mathcal{T} \text{ if } \mathsf{jones}_{k-1}(f') \text{ is not defined} \\ \top_\mathcal{T} \text{ if } \mathsf{jones}_{k-1}(f') = j \text{ and } \mathsf{floor}(i/j) \text{ is odd} \\ \bot_\mathcal{T} \text{ if } \mathsf{jones}_{k-1}(f') = j \text{ and } \mathsf{floor}(i/j) \text{ is even.} \end{cases}$$

Here, floor is the function that rounds down to the next integer.

We now show that we can also encode the if-then-else construct and comparison of Jones encodings of numbers for type $\tau_0 = \bullet$ into tail-recursive HFL, as well as as give a closed definition for objects in $[\![\tau_k]\!]_\mathcal{T}$ that encode the number 0. Moreover, given a set $S \subseteq \{0, \dots, n-1\}$ we can compute the set that encodes the successor of $\mathsf{jones}_0(S)$.

Lemma 5.3.3. *Let $k > 0$. Consider the following, obviously tail-recursive, formulas:*

$$\begin{aligned} \mathsf{ite} &= \lambda(b\colon \tau_0).(x\colon \tau_0).(y\colon \tau_0).\,(b \wedge x) \vee (\neg b \wedge y) \\ \mathsf{zero}_0 &= \mathtt{ff} \\ \mathsf{zero}_{k+1} &= \lambda(x\colon \tau_k).\,\mathtt{ff} \\ \mathsf{gt}_0 &= \lambda(x_1\colon \tau_0).\lambda(x_2\colon \tau_0).\,\langle \mathsf{e}\rangle\big(x_2 \wedge \neg x_1 \wedge [\mathsf{u}]x_1 \to x_2\big) \\ \mathsf{next}_0 &= \lambda(x\colon \tau_0).\,\mathsf{ite}\; x\; \langle \mathsf{d}\rangle\neg x\; [\mathsf{d}]x \\ \mathsf{isZero}_0 &= \lambda(x\colon \tau_0).\,[\mathsf{e}]\neg x \end{aligned}$$

Let $\mathcal{T}$ be an LTS encoding a tiling system $\mathcal{K}$ and let η be an interpretation. Let v be a vertex in $\mathcal{T}$. Then the following hold:

1. *The formulas* $\mathsf{ite}, \mathsf{gt}_0, \mathsf{next}_0, \mathsf{isZero}_0$ *and* zero_k *for $k \geq 0$ are tail recursive.*

2. *If $v \in [\![\psi]\!]^\eta_\mathcal{T}$ then $v \in [\![\mathsf{ite}\,\psi\,\psi_1\,\psi_2]\!]_\mathcal{T}$ if and only if $v \in [\![\psi_1]\!]^\eta_\mathcal{T}$. If $v \notin [\![\varphi]\!]^\eta_\mathcal{T} = \bot_\mathcal{T}$ then $v \in [\![\mathsf{ite}\,\psi\,\psi_1\,\psi_2]\!]_\mathcal{T}$ if and only if $v \in [\![\psi_2]\!]^\eta_\mathcal{T}$.*

3. *$[\![\mathsf{zero}_k]\!]^\eta_\mathcal{T} = 0$ and zero_k is strictly tail recursive for $k \geq 0$.*

4. *If $\mathsf{jones}_0([\![\psi_1]\!]^\eta_\mathcal{T}) = i$ and $\mathsf{jones}_0([\![\psi_2]\!]^\eta_\mathcal{T}) = j$ then $[\![\mathsf{gt}_0\,\psi_1\,\psi_2]\!]^\eta_\mathcal{T}$ is $\top_\mathcal{T}$ if $i < j$ and $\bot_\mathcal{T}$ else.*

5. *If* $\mathsf{jones}_0(\llbracket\psi\rrbracket^{\eta}_{\mathcal{T}}) = i$*, then* $\mathsf{jones}(\llbracket\mathsf{next}_0\,\psi\rrbracket^{\eta}_{\mathcal{T}}) \equiv i + 1 \mod 2^n_1$.

6. $\llbracket\mathsf{isZero}\,\psi\rrbracket^{\eta}_{\mathcal{T}} = \top_{\mathcal{T}}$ *if* $\mathsf{jones}_0(\llbracket\psi\rrbracket^{\eta}_{\mathcal{T}}) = 0$*, otherwise* $\llbracket\mathsf{isZero}\,\psi\rrbracket^{\eta}_{\mathcal{T}} = \bot_{\mathcal{T}}$.

Proof. The claims on tail recursiveness are immediate since no fixpoints occur within the formulas in question. The semantic claims for the first two formulas are straightforward verifications. Regarding the third one, recall that if given two numbers represented in binary, the first one is smaller than the second one if and only if there is a bit that is set in the second one but not the first one, and all bits of lower significance are set in the second number if they are set in the first one. Formula gt_0 encodes this. Let $S_1 = \llbracket\psi_1\rrbracket^{\eta}_{\mathcal{T}}$ and let $S_2 = \llbracket\psi_2\rrbracket^{\eta}_{\mathcal{T}}$. Then $\mathsf{gt}_0\,\psi_1\,\psi_2$ holds at some vertex if there is a vertex i reachable via an e-transition such that this vertex is in the set bound to x_2, i.e, S_2, but not in the set bound to x_2, i.e. S_1, and all vertices reachable from i via a d-transition are in S_2 if they are in S_1. Hence, vertex i is a bit that is set in S_2 but not in S_1, and all bits of lower significance are in S_2 if they are in S_1. It follows that $\mathsf{jones}_0(S_2) > \mathsf{jones}_0(S_1)$ if such a vertex i exists. Since e is the global transition relation, the formula is as claimed in the lemma.

For the fourth claim, recall that a binary number is increased as follows: A bit is set in the incremented number if it is set in the number to be incremented, and there is a bit of lower significance that is not set in the the number to be incremented, or if it is not set in the number to be incremented, but all bits of lower significance are. Let $S = \llbracket\varphi\rrbracket^{\eta}_{\mathcal{T}}$ and let i be a vertex in $\mathcal{T}$. Then, by the first claim, $i \in \llbracket\mathsf{ite}\ x\,(\langle\mathsf{d}\rangle\neg x)\,([\mathsf{d}]x)\rrbracket^{\eta[x\mapsto S]}_{\mathcal{T}}$ if $i \in \llbracket x\rrbracket^{\eta[x\mapsto S]}_{\mathcal{T}}$ and $i \in \llbracket\langle\mathsf{d}\rangle\neg x\rrbracket^{\eta[x\mapsto S]}_{\mathcal{T}}$ or if $i \notin \llbracket x\rrbracket^{\eta[x\mapsto S]}_{\mathcal{T}}$ and $i \in \llbracket[\mathsf{d}]x\rrbracket^{\eta[x\mapsto s]}_{\mathcal{T}}$. By the definition of the transition relation d, the first case is true if $i \in S$ and there is a vertex $j < i$ such that $j \notin S$, and the second case is true if $i \notin S$ and for all $j < i$ we have $j \notin S$. In particular, if $S = \top_{\mathcal{T}}$ then $S' = \emptyset$. Hence, the set $S' = \llbracket\mathsf{next}_0\,\psi\rrbracket^{\eta}_{\mathcal{T}}$ is such that $\mathsf{jones}_0(S') \equiv \mathsf{jones}_0(S) + 1 \mod 2^n_1$.

The claim on isZero_0 is again a straightforward verification. □

In order to extend the previous definitions to orders greater than 0, we have to take into account the peculiarities of Jones' encodings at higher order. In particular, given a number in $0, \dots, 2^n_{k+1} - 1$ there can be several functions in $\llbracket\tau_k\rrbracket_{\mathcal{T}}$ that encode this number. Working with ite, we have to make sure that the semantics of a particular expression are independent of the exact representation of the numbers involved.

Definition 5.3.4. Let $k \geq 0$. We call a function $p \in \llbracket\tau_k \to \bullet\rrbracket_{\mathcal{T}}$ an *arithmetic predicate* if, for all $f_1, f_2 \in \tau_k$ such that $\mathsf{jones}_k(f_1) = \mathsf{jones}_k(f_2)$, we have that $p\,f_1 = p\,f_2$ and, moreover, $p\,f_1 = \top_{\mathcal{T}}$ or $p\,f_1 = \bot_{\mathcal{T}}$.

Note that for $k = 0$, the first part of the criterion to be an arithmetic predicate is trivially satisfied since each number is only represented by one set encoding it. An example for an arithmetic predicate is $\lambda(x : \bullet).\,\mathsf{gt}_0\,\mathsf{zero}_0\,x$.

We are now lifting the definitions from Lemma 5.3.3 to order k. Note that, since the bits of the numbers involved are now sets of vertices or functions themselves, we cannot rely on the transition relation e to quantify over all bits. Instead, we are using additional functions exists_k and forall_k that iterate over all possible bits via fixpoint recursion.

Lemma 5.3.5. *Consider the following formulas for $k > 0$*

$$\begin{aligned}
\mathsf{gt}_k &= \lambda(x_1\colon \tau_k).\lambda(x_2\colon \tau_k).\, \mathsf{exists}_{k-1}\big(\lambda(y\colon \tau_{k-1}).\,(x_2\,y) \wedge \neg(x_1\,y) \wedge \\
&\qquad\qquad \mathsf{forall}_{k-1}(\lambda(z\colon \tau_{k-1}).\,(\mathsf{gt}_{k-1}\,y\,z) \to ((x_1\,z) \to (x_2\,z)))\big)\\
\mathsf{next}_k &= \lambda(x\colon \tau_k).\lambda(y\colon \tau_{k-1}).\,\mathsf{ite}\,(x\,y)\\
&\qquad\qquad \big(\mathsf{exists}_{k-1}\,(\lambda(z_1\colon \tau_{k-1}).\,(\mathsf{gt}_{k-1}\,y\,z_1) \wedge \neg(x\,z_1)\big)\\
&\qquad\qquad \big(\mathsf{forall}_{k-1}\,(\lambda(z_2\colon \tau_{k-1}).\,(\mathsf{gt}_{k-1}\,y\,z_2) \to (x\,z_2)\big)\\
\mathsf{isZero}_k &= \lambda(x\colon \tau_k).\,\mathsf{forall}_{k-1}\,(\lambda(y\colon \tau_{k-1}).\,\mathsf{isZero}_0\,(x\,y))
\end{aligned}$$

and the following formulas for $k \geq 0$

$$\begin{aligned}
\mathsf{exists}_k &= \lambda(p\colon \tau_{k+1}).\,\Big(\big(\mu(X\colon \tau_{k+1}).\,\lambda(x\colon \tau_k).\,(p\,x) \vee X\,(\mathsf{next}_k\,x)\big)\,\mathsf{zero}_k\Big)\\
\mathsf{forall}_k &= \lambda(p\colon \tau_{k+1}).\lambda(x\colon \tau_k).\,\neg\mathsf{exists}(\neg(p\,x))
\end{aligned}$$

Let $k \geq 0$ and let η be some interpretation. Then the following are true:

1. *The formulas are well defined.*

2. *If $\mathsf{jones}_k(\llbracket\psi_1\rrbracket^\eta_\mathcal{T}) = i$ and $\mathsf{jones}_k(\llbracket\psi_2\rrbracket^\eta_\mathcal{T}) = j$ then $\llbracket\mathsf{gt}_k\,\psi_1\,\psi_2\rrbracket^\eta_\mathcal{T} = \top_\mathcal{T}$ if $i < j$ and $\llbracket\mathsf{gt}_k\,\psi_1\,\psi_2\rrbracket^\eta_\mathcal{T} = \bot_\mathcal{T}$ if $i \not< j$.*

3. *If $\mathsf{jones}_k(\llbracket\psi\rrbracket^\eta_\mathcal{T}) = i$ then $\mathsf{jones}_k(\llbracket\mathsf{next}_k\,\psi\rrbracket^\eta_\mathcal{T}) \equiv i + 1 \mod 2^n_{k+1}$.*

4. *$\llbracket\mathsf{isZero}_k\,\psi\rrbracket^\eta_\mathcal{T} = \top_\mathcal{T}$ if $\mathsf{jones}_k(\llbracket\psi\rrbracket^\eta_\mathcal{T}) = 0$, otherwise $\llbracket\mathsf{isZero}_k\,\psi\rrbracket^\eta_\mathcal{T} = \bot_\mathcal{T}$.*

5. *If $\llbracket\psi\rrbracket^\eta_\mathcal{T}$ is an arithmetic predicate, then $\llbracket\mathsf{exists}_k\,\psi\rrbracket^\eta_\mathcal{T} = \top_\mathcal{T}$ if there is $i \in \{0, \dots, 2^n_{k+1} - 1\}$ such that $\llbracket\psi\,\psi'\rrbracket^\eta_\mathcal{T} = \top_\mathcal{T}$ for all ψ' with $\mathsf{jones}_k(\llbracket\psi'\rrbracket^\eta_\mathcal{T}) = i$. Otherwise, $\llbracket\mathsf{exists}_k\,\psi\rrbracket^\eta_\mathcal{T} = \bot_\mathcal{T}$.*

6. *If $\llbracket\psi\rrbracket^\eta_\mathcal{T}$ is an arithmetic predicate, then $\llbracket\mathsf{forall}_k\,\psi\rrbracket^\eta_\mathcal{T} = \top_\mathcal{T}$ if for all $i \in \{0, \dots, 2^n_{k+1} - 1\}$ and for all ψ' $\mathsf{jones}_k(\llbracket\psi'\rrbracket^\eta_\mathcal{T}) = i$ we have that $\llbracket\psi\,\psi'\rrbracket^\eta_\mathcal{T} = \top_\mathcal{T}$. Otherwise, $\llbracket\mathsf{forall}_k\,\psi\rrbracket^\eta_\mathcal{T} = \bot_\mathcal{T}$.*

7. (a) *The formulas gt_k, next_k and isZero_k are tail recursive.*

   (b) *If $\mathsf{tail}^\mathsf{s}(\psi, \mathcal{Y}, \mathsf{N})$ is derivable, then there is a formula ψ' such that $\psi' \equiv \mathsf{exists}_k\,\psi$, the size of ψ' is polynomial in that of $\mathsf{exists}_k\,\psi$, and ψ' is tail recursive and $\mathsf{tail}^\mathsf{s}(\psi', \mathcal{Y}, \mathsf{N})$ is derivable.*

   (c) *If $\mathsf{tail}^\mathsf{s}(\psi, \mathcal{Y}, \mathsf{U})$ is derivable, then there is a formula ψ' such that $\psi' \equiv \mathsf{forall}_k\,\psi$, the size of ψ' is polynomial in that of $\mathsf{exists}_k\,\psi$, and ψ' is tail recursive and $\mathsf{tail}^\mathsf{s}(\psi', \mathcal{Y}, \mathsf{U})$ is derivable.*

Note that e.g., $\mathsf{exists}_k\,\psi$ is not tail recursive if ψ contains free fixpoint variables.

Proof. Regarding well-definedness, note that gt_k, next_k and isZero_k refer to exists_{k-1} and forall_{k-1}, i.e., formulas of one type level less, while exists_k and forall_k refer to gt_k and next_k of the same type level. Hence the mutual recursion is well-defined and grounded in the formulas defined in Lemma 5.3.3.

The rest of the semantic claims is proved by induction on k. For $k = 0$, the formulas gt_k, next_k and isZero_0 are already defined in Lemma 5.3.3. For $k > 0$, they follow the same pattern of bitwise comparison of numbers, respectively incrementation of a number encoded in binary. However, instead of $\langle \mathsf{e} \rangle$ and $[\mathsf{e}]$, we use forall_{k-1} and exists_{k-1}. By the induction hypothesis, the claims for these are already proved. It is then not hard to see that gt_k and next_k are generalizations of the formulas from Lemma 5.3.3. Regarding the claim for exists_k, note that, by fixpoint unfolding and β-reduction,

$$[\![\mathsf{exists}_k\, \psi]\!]^\eta_{\mathcal{T}} = [\![\bigvee_{i \in \mathbb{N}} \psi\, (\mathsf{next}^i_k\, \mathsf{zero}_k)]\!]^\eta_{\mathcal{T}}.$$

Since ψ is an arithmetic predicate, this equals $\top_{\mathcal{T}}$ if $[\![\psi\, (\mathsf{next}^i_k\, \mathsf{zero}_k)]\!]^\eta_{\mathcal{T}} = \top_{\mathcal{T}}$ for at least one $i \in \{0, \dots, 2^n_{k+1} - 1\}$, which is the claim. The claim on forall_k is by simple boolean reasoning.

The claim on tail recursiveness is again by induction on k. The claims for Item 7a are straightforward verifications even if taking exists_{k-1} and forall_{k-1} as defined. Using the claims of Items 7b and 7c, they are immediate. For the proof of Item 7b, consider the β-reduction of $\mathsf{exists}_k\, \psi$, i.e.,

$$\big(\mu(X : \tau_{k+1}).\, \lambda(x : \tau_k).\, (\psi\, x) \vee X(\mathsf{next}_k\, x)\big)\mathsf{zero}_k$$

which is not hard to see to be strictly tail-recursive given the assumptions on ψ. Moreover, it is equivalent to $\mathsf{exists}_k\, \psi$ by invariance of HFL under β-reduction. The proof for Item 7c follows the same pattern. $\square$

5.3.3 Encoding the Tiling Problem

Now that we have the necessary arithmetic machinery in place, we can encode a row of tiles as a function that maps the position of a tile to the singleton set containing the vertex that represents it. Formally, given a row r of length 2^n_k, we say that a formula ψ encodes it if, for all $0 \leq i \leq 2^n_k - 1$, we have that $[\![\psi\, (\mathsf{next}^i_{k-1}\, \mathsf{zero}_{k-1})]\!]^\eta_{\mathcal{T}}$ is the singleton set containing only the vertex j if the ith tile of r is t_j.

Lemma 5.3.6. *Let $k > 0$ and consider the following formulas:*

$$\begin{aligned}
\mathsf{isTile} &= \lambda(x : \tau_0).\, [\mathsf{e}]\Big(x \to \Big(([\mathsf{u}]\neg x) \wedge ([\mathsf{d}]\neg x) \wedge (P_F \vee \langle \mathsf{u} \rangle P_F)\Big)\Big)\\
\mathsf{isRow}_k &= \lambda(r : \tau_k).\, \mathsf{forall}_{k-1}\, (\lambda(x : \tau_{k-1}).\, \mathsf{isTile}(r\, x))\\
\mathsf{init}_k &= \lambda(x : \tau_{k-1}).\, \mathsf{ite}\, (\mathsf{isZero}_{k-1}\, x)\, P_I\, P_{\square}\\
\mathsf{isFinal}_k &= \lambda(r : \tau_k).\, [\mathsf{e}]\big((r\, \mathsf{zero}_{k-1}) \to P_F\big)\\
\mathsf{horiz}_k &= \lambda(r : \tau_k).\, \mathsf{forall}_{k-1}\, \Big(\lambda(x : \tau_{k-1}).\, (\mathsf{isZero}_{k-1}\, (\mathsf{next}_{k-1}\, x)) \vee\\
&\qquad [\mathsf{e}]\big((r\ \ x) \to \langle \mathsf{h} \rangle (r\, (\mathsf{next}_{k-1}\, x))\big)\Big)\\
\mathsf{vert}_k &= \lambda(r_1 : \tau_k).\, \lambda(r_2 : \tau_k).\, \mathsf{forall}_{k-1}\, \Big(\lambda(x : \tau_{k-1}).\, [\mathsf{e}]\big((r_1\, x) \to \langle \mathsf{v} \rangle (r_2\, x)\big)\Big)
\end{aligned}$$

Then the following hold:

1. $[\![\mathsf{isTile}\, \psi]\!]^\eta_{\mathcal{T}} = \top_{\mathcal{T}}$ *if* $[\![\psi]\!]^\eta_{\mathcal{T}}$ *is a singleton set that encodes a tile. Otherwise,* $[\![\mathsf{isTile}\psi]\!]^\eta_{\mathcal{T}} = \bot_{\mathcal{T}}$.

2. $[\![\mathsf{isRow}_k\, \psi]\!]^\eta_{\mathcal{T}} = \top_{\mathcal{T}}$ *if* $[\![\psi]\!]^\eta_{\mathcal{T}}$ *encodes a row. Otherwise,* $[\![\mathsf{isRow}_k\, \psi]\!]^\eta_{\mathcal{T}} = \bot_{\mathcal{T}}$

3. $[\![\mathsf{init}_k]\!]^\eta_{\mathcal{T}}$ *encodes the initial row* $t_I t_\square \cdots t_\square$.

4. Assume that $[\![\psi_1]\!]^\eta_{\mathcal{T}}$ *and* $[\![\psi_2]\!]^\eta_{\mathcal{T}}$ *encode rows. Then*

   (a) $[\![\mathsf{isFinal}\,\psi_1]\!]^\eta_{\mathcal{T}} = \top_{\mathcal{T}}$ *if* $[\![\psi_1]\!]^\eta_{\mathcal{T}}$ *encodes a final row, i.e., a row of the form* $t_F \cdots$. *Otherwise,* $[\![\mathsf{isFinal}\,\psi_1]\!]^\eta_{\mathcal{T}} = \bot_{\mathcal{T}}$.

   (b) $[\![\mathsf{horiz}\,\psi_1]\!]^\eta_{\mathcal{T}} = \top_{\mathcal{T}}$ *if* $[\![\psi_1]\!]^\eta_{\mathcal{T}}$ *is a row where each pair of adjacent tiles matches vertically. Otherwise,* $[\![\mathsf{horiz}\,\psi_1]\!]^\eta_{\mathcal{T}} = \bot_{\mathcal{T}}$.

   (c) $[\![\mathsf{vert}\,\psi_1\,\psi_2]\!]^\eta_{\mathcal{T}} = \top_{\mathcal{T}}$ *if* $[\![\psi_1]\!]^\eta_{\mathcal{T}}$ *and* $[\![\psi_2]\!]^\eta_{\mathcal{T}}$ *encode two rows matching vertically.*

   (d) The functions $\mathsf{isTile}, \mathsf{isRow}_k, I_k, \mathsf{isFinal}_k, \mathsf{horiz}_k$ *and* vert_k *are tail-recursive.*

Proof. The claims on the first four are straightforward verifications. The formula horiz_k tests whether for all $0 \leq i \leq 2^n_{k-1} - 2$, i.e., for each row index apart from the last, it is true that from the unique vertex in the singleton returned by the row at index i, the unique vertex in the singleton returned at index $i+1$ is reachable via an h-transition, which encodes that the tiles encoded by both vertices match horizontally. The formula vert_k does the same by testing whether at each index from 0 to $2^n_{k-1} - 1$, the singletons returned by the formulas encoding the row in question match vertically.

The claims of tail recursiveness are straightforward verifications combined with applications of Lemmas 5.3.3 and 5.3.5. □

The missing piece is now how to generate a successor row given the encoding of a row. Consider the following formula $\mathsf{genSucc}_k$ defined as

$$\lambda(p : \tau_{k+1}).\, \lambda(r_1 : \tau_k).\, \mathsf{exists}_k \big(\lambda(r_2 : \tau_k).\, (\mathsf{isRow}_k\, r_2) \wedge (\mathsf{horiz}_k\, r_2) \wedge (\mathsf{vert}_k\, r_1\, r_2) \wedge (p\, r_2)\big).$$

It consumes an arithmetic predicate of type τ_{k+1} and an object encoding a row. It then iterates over all functions of the form $\mathsf{next}^i_k\, \mathsf{zero}_k$ for $0 \leq i \leq 2^n_k - 1$ and for each of them checks whether it

- encodes a valid row,
- that row has tiles matching horizontally,
- matches the row encoded by r_1 vertically, and
- whether $(p\, r_2)$ holds.

Hence, $\mathsf{genSucc}_k\, (\lambda(r : \tau_k).\, \mathsf{isFinal}_k\, r)\, \mathsf{init}_k$ evaluates whether the initial row has a successor that is final, and $\mathsf{genSucc}_k\, (\lambda(r : \tau_k).\, \mathsf{isFinal}_k\, r)\, (\mathsf{genSucc}^i_k\, \mathsf{init}_k)$ tests whether the initial row has $i+1$ successors such that the last one is a final row. This suggests an encoding of the tiling problem into HFL as a least fixpoint:

$$\big(\mu(P : \tau_{k+1}).\, (\lambda(r_1 : \tau_k).\, (\mathsf{isFinal}_k\, r_1) \vee (\mathsf{genSucc}_k\, P\, r_1)\big)\, \mathsf{init}_k$$

It is not hard to see that this fixpoint is equivalent to the HFL formula

$$\bigvee_{i \in \mathbb{N}} (\mathsf{isFinal}_k\, (\mathsf{genSucc}^i_k\, \mathsf{init}_k))$$

and, hence encodes whether the tiling problem has a successful tiling. However, this formula is neither tail recursive nor strictly tail recursive since P appears as an argument to $\mathsf{genSucc}_k$. However, this can be resolved by β-reducing the application away, which produces the following formula

$$\big(\mu(P\colon \tau_{k+1}).\,(\lambda(r_1\colon \tau_k).\,(\mathsf{isFinal}_k\, r_1) \vee \mathsf{exists}_k\,\big(\lambda(r_2\colon \tau_k).\,\cdots \wedge (P\, r_2)\big)\big)\,\mathsf{init}_k$$

where the omitted part corresponds to the part of the body of $\mathsf{genSucc}_k$ that does not contain p. Note that the body of the fixpoint formula only contains one occurrence of the fixpoint variable P. Hence, it is not hard to see that the conditions of Item 7b of Lemma 5.3.5 are satisfied and we can eliminate the non-tail-recursive use of exists_k in order to obtain an equivalent and strictly tail-recursive formula $\varphi_{\mathcal{K}}^{k+1}$. By the above considerations, given an LTS $\mathcal{T}$ that encodes $\mathcal{K}$ and is of size n where n is at least the number of tiles in $\mathcal{K}$, we have $\mathcal{T}, v \models \varphi_{\mathcal{K}}^{k+1}$ for any v in LTS if and only if the order-k corridor tiling problem of size n for $\mathcal{K}$ has a successful tiling.

Theorem 5.3.7. *The model-checking problem of* $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ *is* k*-EXPSPACE-hard in data complexity for* $k \geq 0$*. More precisely, the problem of deciding whether a given pointed LTS* $\mathcal{T}, v$ *satisfies* $\mathcal{T}, v \models \varphi_{\mathcal{K}}^{k}$ *is* k*-EXPSPACE hard.*

Proof. Fix $\varphi_{\mathcal{K}}^{k+1}$ for a tiling system such that its order-k corridor tiling problem is k-EXPSPACE hard. By the above considerations, for any pointed LTS $\mathcal{T}, v$ it is k-EXPSPACE hard to decide whether $\mathcal{T}, v \models \varphi_{\mathcal{K}}^{k+1}$, since, if $\mathcal{T}$ encodes $\mathcal{K}$ and has at least as many vertices as $\mathcal{K}$ has tiles, then deciding whether $\mathcal{T}, v \models \varphi_{\mathcal{K}}^{k+1}$ amounts to deciding whether there is a successful tiling for the order-k corridor tiling problem of width n on $\mathcal{K}$. By Theorem 5.3.1, this problem is k-EXPSPACE hard. Hence, model-checking order-$(k+1)$ tail-recursive HFL formulas is k-EXPSPACE hard already in data complexity. □

Corollary 5.3.8. *For all* $k \geq 0$*, the fragment* $\mathrm{HFL}^{k}_{\mathsf{tail}}$ *is strictly less expressive than the fragment* $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$.

Proof. For the sake of contradiction, assume that there is $k \geq 0$ such that $\mathrm{HFL}^{k}_{\mathsf{tail}}$ is equi-expressive to $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$. If $k = 0$ then the result is due to the fact that $\mathrm{HFL}^{k}_{\mathsf{tail}}$ is a fragment of the Modal μ-Calculus, which can only express regular properties. On the other hand, already $\mathrm{HFL}^{1}_{\mathsf{tail}}$ contains formulas that express e.g., uniform inevitability, which is known not to be expressible in the $\mathcal{L}_\mu$ [33] but which is expressed by the following formula

$$\big(\mu(X\colon \bullet \to \bullet).\,\lambda(x\colon \bullet).\,x \vee (X\,[a]x)\big)P$$

which is clearly tail recursive.

If $k > 0$ then consider $\varphi_{\mathcal{K}}^{k+1} \in \mathrm{HFL}^{k+1}_{\mathsf{tail}}$ for a tiling system such that its order-k corridor tiling problem is k-EXPSPACE hard. By Theorem 5.3.7, its model-checking problem is k-EXPSPACE hard already in data complexity. It follows that is not decidable in $(k-1)$-EXPSPACE whether a pointed LTS is in

$$\{(\mathcal{T}, v) \mid \mathcal{T}, v \models \varphi_{\mathcal{K}}^{k+1}\}$$

due to the Space Hierarchy Theorem [82]. On the other hand, by assumption, there is $\psi \in \mathrm{HFL}^{k}_{\mathsf{tail}}$ such that $\psi \equiv \varphi_{\mathcal{K}}^{k+1}$, whence

$$\{(\mathcal{T}, v) \mid \mathcal{T}, v \models \varphi_{\mathcal{K}}^{k+1}\} = \{(\mathcal{T}, v) \mid \mathcal{T}, v \models \psi\}$$

holds. But, by Theorem 5.2.10, membership in the latter class can be decided in $(k-1)$-fold exponential space since $\psi \in \mathrm{HFL}^{k}_{\mathsf{tail}}$. This is a contradiction from which we conclude that $\mathrm{HFL}^{k}_{\mathsf{tail}}$ is strictly less expressive than $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$. $\square$

Remark 5.3.9. Note that the formulas in this section do not make use of the possibility to contain unrestricted, i.e., non-tail-recursive fixpoint definitions of low type order. Rather, every single fixpoint definition is tail recursive, and the rule $(\mathsf{fp}_{\mathsf{F}})$ is not used in a derivation of tail-recursiveness for these formulas. Hence, these formulas are *strictly tail-recursive* and the hardness and separation results follow also for $\mathrm{HFL}^{k}_{\mathsf{s\text{-}tail}}$.

Chapter 6

Fixpoint Alternation

In this chapter we study the behavior of fixpoint alternation for HFL and APKA. We begin in Section 6.1 by defining alternation classes in terms of the number and polarity of priorities of an APKA, mirroring the very successful approach for $\mathcal{L}_\mu$, where it has proven fruitful to define the alternation class of a formula via the number and polarity of priorities of any equivalent PA.

We then investigate the behavior of two classes of APKA where the acceptance condition becomes rather manageable. For order-1 APKA, the acceptance condition exhibits LIFO behavior similar to that seen in the context of FLC. For so-called *simple* APKA, in which the occurrences of fixpoint states on the operand side of an application are restricted, the acceptance condition actually reduces to an ordinary parity condition. We use these behaviors to show strictness of the alternation hierarchies for both classes of automata, i.e., we show that, within their respective classes, adding more priorities strictly increases the expressive power. The result for order-1 APKA is from[1] [17].

Finally, in Section 6.3, we show that, over the class of finite LTS, and for *monadic* formulas of low type order, i.e., for those with just one argument, it is possible to rewrite fixpoint definitions such that they are expressed by a fixpoint of the opposite polarity. This is motivated by research into collapse results. The section closes with a discussion of possible extensions of this approach.

6.1 Alternation Classes

Definition 6.1.1. For $n \geq 1$, define

- Σ^n as the class of all APKA that are equivalent to one with priorities in $\{1, \dots, n\}$ if n is even, respectively to one with priorities in $\{0, \dots, n-1\}$ if n is odd,
- Π^n as the class of all APKA that are equivalent to one with priorities in $\{1, \dots, n\}$ if n is odd, respectively to one with priorities in $\{0, \dots, n-1\}$ if n is even,

[1]In [17] it is actually claimed that the fixpoint alternation hierarchy is strict for the whole of HFL. This is not necessarily false, but the result so far only holds to the extent presented here. The automaton model in [17] actually fails to capture full HFL.

- Σ^n_k as the class of all APKA of order at most k that are equivalent to one of order k or less and with priorities in $\{1, \dots, n\}$ if n is even, respectively to one with priorities in $\{0, \dots, n-1\}$ if n is odd,
- Π^n_k as the class of all APKA of order at most k that are equivalent to one of order k or less and with priorities in $\{1, \dots, n\}$ if n is odd, respectively to one with priorities in $\{0, \dots, n-1\}$ if n is even.

Note that Σ^n_k is not necessarily the same as Σ^n restricted to APKA of order k or less. The interaction of type-theoretic order and the number of priorities is currently not well-understood. It is possible that an APKA in Σ^n_k is equivalent to one with higher order, but less priorities, in which case the simple restriction of Σ^n to APKA of a certain order would yield much larger alternation classes than the definition above. Also note that these classes subsume the classes Σ^n_0 and Π^n_0 defined in Section 2.2.6.

For an HFL-formula φ, we say that it belongs to some alternation class if it is equivalent to an APKA in the class. This induces an alternation hierarchy on HFL.

Observation 6.1.2. The following clearly hold for all $n \geq 1$:

- $\Sigma^n \subseteq \Sigma^{n+1}$,
- $\Pi^n \subseteq \Pi^{n+1}$,
- $\Sigma^n \subseteq \Pi^{n+1}$,
- $\Pi^n \subseteq \Sigma^{n+1}$,
- If $\mathcal{A} \in \Sigma^n$, then $\overline{\mathcal{A}} \in \Pi^n$.
- If $\mathcal{A} \in \Pi^n$, then $\overline{\mathcal{A}} \in \Sigma^n$.

Moreover, the same inclusions hold if restricted to a given type-theoretic order, e.g., $\Sigma^n_k \subseteq \Sigma^{n+1}_k$ holds.

The above inclusions allow to speak of an *alternation hierarchy.* We say that the hierarchy is *strict* if infinitely many of the inclusions above are strict. We say that the hierarchy *collapses* for a given fragment of APKA or HFL, or over a given class of LTS, if for some n, all of the inclusions become improper. As we have seen in the context of $\mathcal{L}_\mu$, the alternation hierarchy for $\mathcal{L}_\mu$ is strict over the class of all structures and over several subclasses, but collapses over several other. See [41] for an overview.

What is obviously missing from the above definitions is a characterization of alternation-free behavior in the context of HFL. Since it is not straightforward to generalize the definition of e.g., weak automata to APKA, this problem is postponed to further research.

Remark 6.1.3. We will assume without further mention that automata in Σ^n and Π^n actually do have priorities $1, \dots, n$ or $0, \dots, n-1$ as in Definition 6.1.1, and similarly for the classes restricted to certain type levels.

6.2 Strictness Results over Infinite Trees

6.2.1 Strictness for HFL^1

In this section, we show that the fixpoint alternation hierarchy is strict when restricted to order-1 APKA. The reason for this is that for order-1 APKA, the acceptance condition is not more complicated than that for FLC, or, equivalently, a stair-parity condition. Hence, whether a play of the APKA acceptance game is winning can be expressed in an order-1 APKA again. This allows us to adapt Arnold's proof, respectively that for the strictness of the FLC alternation hierarchy, to order-1 APKA and, hence, to HFL^1.

Properties of Order-1 APKA

We begin by analyzing the behavior of order-1 APKA. Remember that, if $C = (\_\,, (Q, e), \Gamma)$ is a configuration in a play of the APKA acceptance game, and $C' = (\_\,, (\delta(Q, e), \varepsilon)$ is the configuration following it, then C' is called the configuration where e was defined, and e is the environment associated to C'.

Lemma 6.2.1. *Let $\mathcal{A}$ be an* APKA *of order at most* 1 *and let $(C_i)_{i\in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}$ over some LTS. Let $C = (\_\,, (Q, e), \Gamma)$ be a fixpoint configuration in this play. Then the environment in the configuration following C is of the form $(f_1^Q \mapsto (\chi_1, e), \dots, f_{k_Q}^Q \mapsto (\chi_{k_Q}, e), \_)$, i.e., it maps all variables to closures in e.*

Proof. If C is the initial configuration of the play, then there is nothing to prove. Otherwise, let C' be the last configuration before C such that its argument stack is empty. Since this is the case for the configuration in which e was created, such a configuration must exist. If C is the configuration immediately after C', we are done. Otherwise, C' is an application configuration. Since $\mathcal{A}$ is of order 1, the operator is of type $\bullet \to \cdots \to \bullet$ and the operand is of type $\bullet$. In fact, all configurations strictly between C' and C are application configurations, and their operands are of type $\bullet$. This is because the only possible configurations with closures not of ground type are application configurations, lambda variable configurations, and fixpoint variable configurations. The next fixpoint configuration after C' is C, for otherwise the configuration immediately after such a fixpoint configuration would have an empty stack, contradicting the definition of C' as the last such configuration. On the other hand, since $\mathcal{A}$ is of order 1, all lambda variables are of ground type and no lambda variable can occur between C' and C. But if all configurations strictly between C' and C are application configurations, their environment component never changes and is equal to that of C, namely e. Moreover, all closures on the stack have the same environment component. The claim of the lemma then follows. $\square$

Lemma 6.2.1 justifies the following definition.

Definition 6.2.2. Let $\mathcal{A}$ be an APKA of order at most 1 and let $(C_i)_{i\in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}$ over some LTS. Let $e' \neq e^0$ be an environment in this play and let $(\_\,, (Q, e), \_)$ be the fixpoint configuration immediately preceding the configuration where e' was defined. Then e is called the *parent environment* of e'.

Hence, all variables in an environment that is not e^0 point to closures in the parent environment. Moreover, since the environment component of configurations in a play of the acceptance game changes only at fixpoint configurations and at lambda variable configurations, for an APKA of order 1 it changes in a very controlled way: In the first case, the environment component passes to a new environment such that the previous environment is the parent environment of the new one. In the second case, it passes from an environment to its parent environment. This is also reflected in the structure of the unfolding tree of a play:

Lemma 6.2.3. *Let $\mathcal{A}$ be an* APKA *of order at most* 1 *and let $\pi = (C_i)_{i \in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}$ over some LTS. Let $C = (\_\,, (Q, e), \_)$ be a fixpoint configuration and let e' be the environment associated to the configuration following C. Let C' be the configuration where e, the parent environment of e', was created. Then the node labeled by C is a descendant of the node labeled by C' in the unfolding tree of π.*

Proof. By Lemma 4.2.26, all nodes labeled by configurations in e are descendants of the node labeled by C'. □

The above considerations, together with the fact that all lambda variables are of ground type, yield a necessary and sufficient criterion on when a fixpoint configuration is on the infinite path in a run:

Lemma 6.2.4. *Let $\pi = (C_i)_{i \in \mathbb{N}}$ be a play in the acceptance game for an* APKA *of order at most* 1 *and let $\mathbb{T}^\pi$ be the unfolding tree associated with this play. Let C be a fixpoint configuration in this play and let e be the environment associated to C. Then C is on the infinite path in $\mathbb{T}^\pi$ if and only if during the play there is no configuration of the form $(\_\,, (f, e), \_)$.*

Proof. Note that, since $\mathcal{A}$ is of order at most 1, its generalized unfolding tree reduced down to type level 1 is the same as its unfolding tree. By Lemma 4.2.25, branching in any generalized unfolding tree is at most binary in nodes labeled by application configurations where the operand is of ground type. Since $\mathcal{A}$ is of order at most 1, the whole unfolding tree of the play has at most binary branching. Moreover, by Lemma 4.2.31, if a node that has both an operator subtree and an operand subtree, the operator subtree is finite and, hence, contains no infinite path.

Let t be the node labeled by C. Assume that in the play, there is a configuration of the form $(\_\,, (f, e), \_)$. By Lemma 4.2.26, the node u labeled by this configuration is a descendant of t. Moreover, t is, in turn, a descendant of the node u' labeled by $\mathsf{bnode}_f(e)$, which must have an operand son since the closure (f, e) occurs in a configuration during the play. Hence, the operator subtree of u' is finite, and t, which is contained in it, is not on the infinite path.

Conversely, assume that no configuration of the form $(\_\,, (f, e), \_)$ occurs during the play. Let $\mathcal{E}_C$ denote the set of environments such that e' is in it if and only if there is a finite sequence of environments $e' = e_1, \ldots, e_n = e$ such that e_{i+1} is the parent environment of e_i for all $1 \leq i \leq n - 1$. Note that $e \in \mathcal{E}_C$ and that $\mathcal{E}_C$ contains an environment e' if and only if it also contains all environments that have e' as a parent environment, and, if $e' \neq e$, also the parent environment of e'.

We now claim that, starting from the configuration following C, i.e., the configuration where e is created, the environment component of all further configurations is from $\mathcal{E}_C$. Certainly, this is true for said configuration where e is created. Now assume that we have proved this for all configurations up to some configuration C'. Let $e' \in \mathcal{E}_C$ be the environment component of C'. If C is neither a fixpoint configuration nor a lambda variable configuration, the environment component of the configuration following C' is also e' and there is nothing to prove. If C' is a fixpoint variable configuration, then its successor will have a newly created environment component e''. Note that the parent environment of e'' is e', so $e'' \in \mathcal{E}_C$. Finally, if C' is a lambda variable configuration, note that $e' \neq e$ by assumption. Hence, by Lemma 6.2.1, this variable points to a closure in the parent environment of e', which is also in $\mathcal{E}_C$. Hence, the claim is proved.

By Lemma 4.2.24, all nodes in $\mathbb{T}^\pi$ that are labeled by a configuration with environment component e' are descendants of the node labeled by the configuration where e' was created, and this node, in turn, is the unique son of the environment labeled by the fixpoint configuration associated to e'. We now claim that all configurations with an environment component in $\mathcal{E}_C$ label descendants of the node labeled by C: Let C' be such a configuration, and let $e' \in \mathcal{E}_C$ be its environment component. Then there is a finite sequence of environments $e' = e_1, \ldots, e_n = e$ such that e_{i+1} is the parent environment of e_i for all $1 \leq i \leq n-1$. If $e = e'$, there is nothing to prove due to Lemma 4.2.24. Otherwise, the node labeled by C' is a descendant of the node labeled by the configuration where the e' was created. Moreover, the fixpoint configuration associated to e' is the direct predecessor of the node where $e' = e_1$ was created. Since e_2 is the parent environment of e_1, this fixpoint configuration is in e_2. By repeating this argument $n-1$ times, we obtain that the node labeled by C' is a descendant of the node labeled by C.

Hence, all nodes labeled by configurations after C are descendants of the node labeled by C'. Since the play is infinite by assumption, the tree below the node labeled by C is infinite and has at most binary branching due to being a subtree of $\mathbb{T}^\pi$. Hence, due to Kőnig's Lemma, this subtree contains an infinite path. Since this is a subtree of $\mathbb{T}^\pi$, it must be the unique infinite path of $\mathbb{T}^\pi$, whence the node labeled by C is on that infinite path.

□

Definition 6.2.5. Let $(C_i)_{i\in\mathbb{N}}$ be a play in the acceptance game for an APKA of order at most 1. Let C be a fixpoint configuration in this play and let e be the environment associated to C. If there is a configuration $C_i = (\_, (f, e), \_)$, then we call e *closed* in all subsequent configurations.

Note that the notion of a closed environment is relative to a configuration in the play, i.e., an environment can be not closed for a number of configurations after its creation, but still be closed later. Naturally, an environment that is closed in one configuration is closed in all subsequent configurations. Finally, after the conclusion of an infinite play of the acceptance game for an APKA of order at most 1, a fixpoint configuration labels a node on the infinite path of the unfolding tree of the play if and only if its associated environment was never closed during the play.

The definition of parent environments and closed environments also lays bare the LIFO nature of the acceptance game for order-1 APKA. Since occurrences of fixpoint configurations are coupled to the creation of new environments, and since the parent environment of another environment can only be closed once the other environment is closed, a play of an order-1 APKA generates an oscillating sequence of unclosed environments such that, eventually, the initial sequence will stay unclosed indefinitely. This mirrors closely the behavior of FLC [62] and can be considered to be a stair-parity condition [66]. The connection to FLC is not surprising, since FLC is essentially the fragment of order-1 HFL obtained by restriction to types of arity 1.

Encoding a Run Into a Tree

Following the pattern in Arnold's proof of the strictness of the $\mathcal{L}_\mu$ alternation hierarchy (cf. Section 2.2.6 for a sketch), we now encode the game graph of the acceptance game of an APKA of order at most 1 over a fully infinite binary tree into a fully infinite binary tree again.

For each $n \geq 1$, define a set of propositions $\mathcal{P}_n = \{N, U, V, T, F, P_1, \dots, P_n\}$. Consider the class of fully infinite binary trees with labels in $\mathcal{P}_n$, and where the single transition is denoted by a. Given such a tree $\mathbb{T}$, and an APKA $\mathcal{A}$ of order 1, such that $\mathcal{A} \in \Sigma^n_1$ or $\mathcal{A} \in \Pi^n_1$, the game graph of the acceptance game of $\mathcal{A}$ over $\mathbb{T}$, starting from its root ε is almost a fully infinite binary tree itself. We now define a tree $T(\mathcal{A}, \mathbb{T})$ with labels in $\mathcal{P}_n$ that encodes enough of this game graph to make the winner of the acceptance game definable by an APKA of order 1 of the same alternation class as $\mathcal{A}$ again. Each reachable configuration in the game graph of the acceptance game induces at least one node in $T(\mathcal{A}, \mathbb{T})$, but some configurations can induce more than one. We inductively describe which configurations induce a node, and how these nodes are labeled. The intuition is that all nodes induced by configurations where $\mathcal{V}$ can make a decision, for example disjunction configurations, are labeled by N (for nondeterministic), nodes induced by configurations where $\mathcal{S}$ can make a decision are labeled by U (for universal), nodes induced by lambda variable configurations are labeled by V, nodes induced by configurations where $\mathcal{V}$ or $\mathcal{S}$ outright wins the game are labeled by T, respectively F, and nodes induced by fixpoint configurations are labeled by P_i, where i is the priority of the fixpoint variable if the automaton has priorities in $\{1, \dots, n\}$ and $i-1$ is the priority of the fixpoint variable if the automaton has priorities in $\{0, \dots, n-1\}$. Nodes induced by application configurations are also labeled by N for the sake of simplicity.

The root of $T(\mathcal{A}, \mathbb{T})$ is induced by the initial configuration of the acceptance game of $\mathcal{A}$ over $\mathbb{T}$, i.e., $(\varepsilon, (Q_I, e^0), \varepsilon)$. The labeling of a node induced by a configuration $C = (t, (\chi, e), \Gamma)$ depends on the type of χ:

- If $\chi = P$ then the node induced by C is labeled by T if $\mathbb{T}, t \models P$ and it is labeled by F if $\mathbb{T}, t \not\models P$. Conversely, if $\chi = \overline{P}$, then the node induced by C is labeled by T if $\mathbb{T}, t \not\models P$ and by F if $\mathbb{T}, t \models P$. Both the left and the right successor of the node is induced by the C again.

- If $\chi = \chi_1 \vee \chi_2$ or $\chi = \chi_1 \wedge \chi_2$, the node is labeled by N, respectively by U. The left successor of the node is induced by the configuration $(t, (\chi_1, e), \Gamma)$ and the right successor of the node is induced by the configuration $(t, (\chi_2, e), \Gamma)$.

- If $\chi = \langle a \rangle \chi'$ or $\chi = [a]\chi$, then the node is labeled by N, respectively by U. The left successor of the node is induced by the configuration $(t0, (\chi', e), \Gamma)$, while the right successor of the node is induced by the configuration $(t1, (\chi', e), \Gamma)$.

- if $\chi = \chi_1\, \chi_2$, then the node is labeled by N and both its left and the right successor are induced by the successor configuration of C.

- If $\chi = f$, then the node is labeled by V and both its left and the right successor are induced by the successor configuration of C.
- If $\chi = Q$ such that $\Delta(Q) = i$, then the node is labeled by P_i, if the automaton has priorities in $\{1, \dots, n\}$ and P_{i+1}, if the automaton has priorities in $\{0, \dots, n-1\}$. In both cases, both its left and the right successor are induced by the successor configuration of C.

Note that, in order to make the tree fully infinite and binary, at nodes induced by a configuration that only has one successor, we use two copies of that successor as the configurations inducing the two successor nodes. At nodes induced by a configuration where the play ends, we repeat that configuration indefinitely to make this subtree infinite. Also note that the above definition requires the root of $T(\mathcal{A}, \mathbb{T})$ to be labeled by P_i, where i, respectively $i-1$, is the priority of the initial state of $\mathcal{A}$.

Automata That are Hard for Alternation Classes

Definition 6.2.6. Let the order-1 APKA $\mathcal{A}_n^\Sigma = (\mathcal{Q}, \Delta_n^\Sigma, I, \delta_\Sigma, (\tau_Q)_{Q \in \mathcal{Q}})$ and $\mathcal{A}_n^\Pi = (\mathcal{Q}, \Delta_n^\Pi, I, \delta_\Pi, (\tau_Q)_{Q \in \mathcal{Q}})$ be defined via

- $\mathcal{Q} = \{I, Q_1, \dots, Q_n, O\}$,
- $$\Delta_n^\Sigma(Q) = \begin{cases} i \text{ if } Q = Q_i \text{ and } n \text{ is even} \\ i-1 \text{ if } Q = Q_i \text{ and } n \text{ is odd} \\ \Delta_n^\Sigma(Q_1) \text{ if } Q = I \text{ or } Q = O \end{cases}$$
$$\Delta_n^\Pi(Q) = \begin{cases} i \text{ if } Q = Q_i \text{ and } n \text{ is odd} \\ i-1 \text{ if } Q = Q_i \text{ and } n \text{ is even} \\ \Delta_n^\Pi(Q_1) \text{ if } Q = I \text{ or } Q = O \end{cases}$$
- $\delta(I) = O\,T$ and $\delta(Q_1) = O\,f_1$ and $\delta_{Q_i} = Q_{i-1}\,f_i$ for $1 < i \leq n$, and $\delta(O) = T \vee (\overline{F} \wedge \chi)$ where
$$\chi = \big(\overline{N} \vee \langle a \rangle (O\,f)\big) \wedge \big(\overline{U} \vee [a](O\,f)\big) \wedge \big(\overline{V} \vee \langle a \rangle f\big) \\ \wedge \big(\overline{P_1} \vee \langle a \rangle (Q_1\,(O\,f))\big) \wedge \dots \wedge \big(\overline{P_n} \vee \langle a \rangle (Q_n\,(O\,f))\big)$$
- $\tau_I = \bullet$ and $\tau_{Q_1} = \dots = \tau_{Q_n} = \tau_O = \bullet \to \bullet$.

Note that each fixpoint state Q_i has a single lambda variable f_i, while state O has a single lambda variable f.

Clearly, $\mathcal{A}_n^\Sigma$ is in Σ_1^n, and $\mathcal{A}_n^\Pi$ is in Π_1^n.

We analyze $\mathcal{A}_n^\Sigma$ and $\mathcal{A}_n^\Pi$ informally. Considering the definition of $\mathcal{A}_n^\Sigma$ and $\mathcal{A}_n^\Pi$, note that the values of the lambda variables in environments are quite simple. In the initial state, O is called with argument T, and in the fixpoint states $Q_1, \dots, Q_n$, O is always called with the variable of the fixpoint state itself as an argument. These states are called themselves only with argument $O\,f$. It follows that the value of a lambda variable in an environment e is either

- $(T, \_)$ in the special case of the second[2] environment created,
- (f', e') where e' is the parent environment of e and f' is the single variable defined by it, or
- $(O\,f, e')$ where, again, e' is the parent environment of e.

Upon reaching a lambda variable configuration, what happens depends on the value of said variable. If it falls into the first case, the play in question will end after another step. If it falls into the second case, variable lookup will continue. Since variables in APKA of order 1 point to closures in their parent environment, this cannot be chained indefinitely, and eventually, a chain of variable configurations of the second kind will end in one of the first kind, or the third kind. Together with the definition of the automata, which are either straightforward calls of the state O, or boolean and modal combinations of such calls and of lambda variable lookups, it is clear that any play of the acceptance game for either automaton repeatedly goes through fixpoint configurations of state O. This motivates the following definition:

Definition 6.2.7. Let $(C_i)_{i \in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over a tree $T(\mathcal{A}, \mathbb{T})$ where $\mathcal{A} \in \Sigma_1^n$, respectively $\mathcal{A} \in \Pi_1^n$. A *round* is a sequence of configurations, starting in a configuration of the form $(\_, (O, \_), \_)$ and contains all configurations up to, but not including the next configuration of this form, if it exists. An environment is *associated to a round* if it is created in a configuration in that round. An environment associated to a round is called the *lower* environment of this round if it is the environment with highest index associated to that round, and it is called the *higher* environment of this round if it is the environment with the lowest index associated to that round[3].

A round is called *inactive* if its lower environment is closed, and it is called *closed* if its upper environment is closed. It is called *unclosed* if it is not closed.

A round r is the *parent round* of another round r', or just the parent, if the initial configuration $(\_, (O, e), \_)$ of r' is such that e is associated to r.

A round is called an F-round, T-round, N-round, U-round, V-round or an i-round if the first configuration $(t, (O, \_), \_)$ in it is such that $\mathcal{L}(t)$ is the singleton containing F, T, N, U, V or P_i, respectively.

Finally, the first two configurations $(\varepsilon, (I, e^0), \varepsilon)$ and $(\varepsilon, (O\,T, e_1), \varepsilon)$ are considered to be part of the first round in the run, which is necessarily an i-round for some i. This round is called the *initial round.*

Note that a round can be parent round to several other rounds, or to none, even if the play continues after it. In particular, a round is not necessarily the parent of the round of configurations that follow it in the sequence of the play. Also, a round that is closed is necessarily inactive, even for rounds with several associated environments. Similarly to the case of closed environments, the notion of a closed or inactive round is to be understood with respect to a configuration in a play of the acceptance game. Hence, a round can be not inactive, inactive, and finally closed during the same play, but an inactive round can only either stay inactive or become closed in a later configuration, and a closed round is closed in all subsequent

[2]Remember that the unfolding of I already creates an environment.

[3]We show in Lemma 6.2.8 that, with one exception, all rounds have at most two environments associated to them.

configurations. Finally, due to the nature of closed environments, a round is closed if its parent is closed.

The idea here is that a play of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over a tree $T(\mathcal{A}, \mathbb{T})$ encoding the game graph of some APKA $\mathcal{A}$ over some tree $\mathbb{T}$ induces a play of $\mathcal{A}$ over $\mathbb{T}$. Each round in the play for $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ manages the behavior of a single configuration in the play of $\mathcal{A}$. In N-rounds and U-rounds, $\mathcal{V}$, respectively $\mathcal{S}$, chooses the further development of this play. In i-rounds, a fixpoint configuration of priority i or $i-1$, depending on n and the alternation class in question, is visited, and $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$ emulate this by visiting state Q_i. In V-rounds, a variable configuration is visited, signaling the closing of the newest unclosed environment in the play for $\mathcal{A}$. In the play for $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$, all rounds up to and including the next i-round are closed, correctly mirroring the acceptance condition.

Note that in an F-round and a T-round, the play will end since one of the players has a winning strategy, and in an X-round, for $X = N, U, V, P_1, \ldots, P_n$, it is not hard to see that $\mathcal{S}$ will force the play to go into the conjunct of the transition relation of O that starts with $\overline{X} \vee \ldots$. Without loss of generality, we assume from now on that he does so.

Lemma 6.2.8. *Let $(C_i)_{i \in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over a tree $T(\mathcal{A}, \mathbb{T})$, where $\mathcal{A} \in \Sigma_1^n$, respectively $\mathcal{A} \in \Pi_1^n$. The following hold for all rounds except the initial one:*

- *The upper environment of a round r is associated to the occurrence of O in the round and either binds f to*
  - *$(O\, f_i, e')$, where e' is the lower environment of the parent round r' of r if r' is an i-round for some i and not inactive in the first configuration of r, or binds f to*
  - *(f, e') where e' is the upper environment of the parent round r' of r if r' is not an i-round or an i-round that is inactive in the first configuration of r.*
- *There are two environments associated to an i-round, namely the upper environment e associated to the occurrence of O, and the lower environment associated to an occurrence of Q_i. The lower environment binds f_i to $(O\, f, e)$.*
- *There is one environment associated to N-rounds and U-rounds, which is simultaneously the upper and the lower environment of that round.*
- *There is one environment associated to V-rounds. Moreover, after a V-round r the play either ends, or there is a sequence $r = r_1, \ldots, r_k$ such that r_{j+1} is the parent of r_j for all $1 \leq j \leq k-1$ and r_j is an N-round or a V-round for all $1 \leq j \leq k-1$, and r_k is an i-round. After the conclusion of r, all the r_j are closed, and r_k is inactive.*

Moreover, during each round, including the initial one, there is exactly one transition unless the round is the last one, i.e., the play ends during that round. The target of the transition is chosen by $\mathcal{S}$ in U-rounds and by $\mathcal{V}$ in all other rounds.

Proof. The proof is by induction over the length of the play. The induction hypothesis is that the first configuration in a round r that is not the initial one is of the form $(t, (O, e), (f', e))$ where f' is the unique variable defined in e and:

- e is the lower environment of the parent round r' of r in case that r' is an i-round that is not inactive, or
- e is the upper environment of the parent round r' of r in case r' is not an i-round, or inactive.

Moreover, assume that the lemma has been proved for all previous rounds, i.e., all rounds containing configurations of lower index.

We first investigate the initial round. Since it is necessarily an i-round, where i is the priority of the initial state of $\mathcal{A}$, it proceeds through the configurations (with new environments displayed to the right of the configuration where they are created)

$$\begin{array}{rl}
(\varepsilon, (I, e^0), \varepsilon) & \\
(\varepsilon, (O\,T, e_1), \varepsilon) & \quad e_1 = (\emptyset, 1) \\
\varepsilon, (O, e_1), (T, e_1) & \\
(\varepsilon, (T \vee (\overline{F} \wedge \chi), e_2), \varepsilon) & \quad e_2 = (f \mapsto (T, e_1), 2) \\
(\varepsilon, (\overline{F} \wedge \chi, e_2), \varepsilon) & \\
(\varepsilon(\chi, e_2), \varepsilon) & \\
(\varepsilon(\overline{P_i} \vee \langle a \rangle Q_i\,(O\,f), e_2), \varepsilon) & \\
(\varepsilon, (\langle a \rangle Q_i\,(O\,f), e_2), \varepsilon) & \\
(j, (Q_i\,(O\,f), e_2), \varepsilon) & \\
(j, (Q_i, e_2), (O\,f, e_2)) & \\
(j, (O\,f_i, e_3), \varepsilon) & \quad e_3 = (f_i \mapsto (O\,f, e_2), 3) \\
(j, (O, e_3), (f_i, e_3)) &
\end{array}$$

where χ is as in the definition of the automata, j is either 0 or 1 and denotes the target of the unique transition during the round, which is chosen by $\mathcal{V}$. The last configuration shown is the first configuration of the next round, of which the initial round is the parent round, since e_3 is associated to the initial round. Note that e_3 is the lower environment of the initial round, whence this first configuration of the next round satisfies the induction invariant.

Now assume that the first configuration in a round r that is not the initial one is of the form $(t, (O, e), (f', e))$ where f' is the unique variable defined in e and:

- e is the lower environment of the parent round r' of r in case that r' is an i-round that is not inactive, or
- e is the upper environment of the parent round r' of r in case r' is not an i-round, or inactive.

Moreover, assume that the lemma has been proved for all previous rounds, i.e., all rounds containing configurations of lower index.

The first move of r is towards $(t, (T \vee (\overline{F} \wedge \chi), e'), \varepsilon)$ where $e' = (f \mapsto (f', e), \_)$ and χ is as in the definition of either automaton. Note that e' is the upper environment of r and is as claimed in the lemma by the induction hypothesis.

The rest of the round depends on its type. T-rounds and F-rounds end after one, respectively two further moves. X-rounds that are neither T-rounds nor F-rounds end up in the configuration $(t, (\overline{X} \vee \zeta, e'), \varepsilon)$ and then in $(t, (\zeta, e'), \varepsilon)$, where ζ depends on the type of the round.

- In i-rounds, ζ is $\langle a\rangle(Q_i\,(O\,f))$. Hence, $\mathcal{V}$ chooses $j \in \{0,1\}$ and the play continues to $(tj, (Q_i\,(O\,f), e'), \varepsilon)$ and further through

$$\begin{array}{ll} (tj, (Q_i, e)(O\,f, e')) & \\ \quad (tj(O\,f_i, e''), \varepsilon) & e'' = (f_i \mapsto (O\,f, e'), \_\,) \\ (tj, (O, e''), (f_i, e'')) & \end{array}$$

  where e'' is the lower environment of the round and is as claimed in the lemma.

- In N-rounds and U-rounds, ζ is $\langle a\rangle(O\,f)$, respectively $[a](O\,f)$. Hence, $\mathcal{V}$, respectively $\mathcal{S}$, chooses $l \in \{0,1\}$ and the play continues through $(tl, (O\,f, e'), \varepsilon)$ to $(tl, (O, e'), (f, e'))$, which is the first configuration of the next round. Clearly, this configuration satisfies the induction hypothesis, the round described has exactly one transition and one environment associated to it, namely e', which is both the upper and the lower environment of the round.

- In V-rounds, ζ is $\langle a\rangle f$. Hence, $\mathcal{V}$ chooses $l \in \{0,1\}$ and the play continues to $C' = (tl, (f, e'), \varepsilon)$. Reading this configuration closes the round. We now construct a sequence of rounds. Let $r_1 = r$ and note that, for $j = 1$, we have that C' is of the form $(tl, (f_j, e_j), \varepsilon)$ where e_j is the upper environment of r_j. Given r_j, there are three possibilities, depending on the parent round r_{j+1} of r_j:

  - If r_{j+1} is the initial round, then $\mathsf{lookup}(f_j, e_j) = (f_i, e_3)$. Comparing this to the analysis of the initial round, it is easy to see that the play ends in three moves.
  - If r_{j+1} is an N-round, a U-round, or an inactive i-round, then we have that $\mathsf{lookup}(f_j, e_j)$ is a closure of the form (f, e'') that is again the upper environment of r_{j+1}. Moreover, reading the next configuration $(tl, (f, e''), \varepsilon)$ closes r_{j+1}. In this case, let $(f, e'') = (f_{j+1}, e_{j+1})$ and continue to construct r_{j+2}.
  - If r_{j+1} is an active i-round, then $\mathsf{lookup}(f_j, e_j) = (f_i, e'')$ where e'' is the lower environment of r_{j+1}. In this case, let $k = j + 1$. Moving from $(tl, (f_i, e''), \varepsilon)$ to $(tl, (O, f, e'''), \varepsilon)$, where e''' is the upper environment of r_{j+1}, makes r_{j+1} inactive. Note that this last configuration is as in the induction hypothesis

  Since r_{j+1} is the parent round of r_j, this chain of rounds can only have finite length. It follows that either r is the last round, since the chain ends in the first case and the play ends, or the chain of N-rounds and U-rounds ends in an i-round that is now inactive, and all other rounds in the chain are closed.

In either case, the round has exactly one transition, unless it is a T-round or an F-round, and the target of the transition is chosen by $\mathcal{V}$, unless the round is a U-round.

The claim of the lemma then follows by induction over the length of the play. □

Lemma 6.2.9. *Let $\mathcal{A} \in \Sigma_1^n$ or $\mathcal{A} \in \Pi_1^n$ be an* APKA *of order* 1*, and let $\mathbb{T}$ be a tree with labels in $\mathcal{P}_n$. Let $(C_i)_{i\in I}$ be a play of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over $T(\mathcal{A}, \mathbb{T})$. Let $(t_i)_{i\in I'}$ be the* *sequence of nodes in $T(\mathcal{A}, \mathbb{T})$ visited during this play. Let $(C'_i)_{i \in I'}$ be the sequence of configurations in the play for $\mathcal{A}$ such that C'_i induces t_i. Then $\mathcal{V}$ wins the play $(C_i)_{i \in I}$ if and only if she wins the play $(C'_i)_{i \in I'}$.*

Proof. For the sake of simplicity, we only treat the case of $\mathcal{A}^{\Sigma}_n$ and even n. The case for odd n is by an additional shift of the priorities involved by 1, and the case for $\mathcal{A}^{\Pi}_n$ is completely symmetric. The proof is by induction over the length of the acceptance game of $\mathcal{A}^{\Sigma}_n$ over $T(\mathcal{A}, \mathbb{T})$. The induction invariant is the following: At the beginning of each round after the first one, the play of the acceptance game of $\mathcal{A}^{\Sigma}_n$ is in a configuration of the form $C = (t, (O, \_), \_)$ such that, if C' is the configuration that induces t, then:

- C belongs to a round r, and
  - the set of rounds that are not closed in the acceptance game of $\mathcal{A}^{\Sigma}_n$ is $\{r^1_1, \ldots, r^1_{k_1}, \ldots, r^m_1, \ldots, r^m_{k_m} = r\}$,
  - for all $1 \leq j \leq m$ and all $1 \leq j' \leq k_j - 1$, we have that $r^j_{j'}$ is the parent round of $r^j_{j'+1}$,
  - for all $1 \leq j \leq m - 1$, we have that $r^j_{k_j}$ is the parent round of r^{j+1}_1,
  - for all $1 \leq j \leq m$, we have that r^j_1 is an i-round for some i,
  - for all $1 \leq j \leq n$ and $1 < j' \leq k_m$, we have that $r^j_{j'}$ is an N-round, a U-round or an inactive i-round for some i.
- The set of environments not closed in C' is $e^0, e_1, \ldots, e_m$ such that e_j is the parent environment of e_{j+1} for all $1 \leq j \leq m - 1$.
- For all $1 \leq j \leq m$ and for all $1 \leq i \leq n$, the fixpoint state in $\mathcal{A}$ of the fixpoint configuration associated to e_j has priority i if and only if r^j_1 is an i-round.

Consider the starting position of the acceptance game of $\mathcal{A}^{\Sigma}_n$, which is $(\varepsilon, (I, e^0), \varepsilon)$, where the node ε is induced by the initial configuration of the play of the acceptance game of $\mathcal{A}$, which is $(\varepsilon, (Q_I, e^0), \varepsilon)$. Let p be the priority of Q_I, the initial state of $\mathcal{A}$. Then $\varepsilon \in T(\mathcal{A}, \mathbb{T})$ is labeled by P_p, and, hence, the initial round in the acceptance game of $\mathcal{A}^{\Sigma}_n$ over $T(\mathcal{A}, \mathbb{T})$ is a p-round. The round proceeds as seen in the proof of Lemma 6.2.8, i.e., $\mathcal{V}$ chooses a successor of ε in $T(\mathcal{A}, \mathbb{T})$. Since both subtrees are isomorphic by definition, $\mathcal{V}$'s choice does not matter: the root of both subtrees is induced by the configuration $(\varepsilon, (\delta(Q_I), e_1), \varepsilon)$. Here, e_1 is the first environment of the play of the acceptance game of $\mathcal{A}$. Hence, at the beginning of the first round after the initial round, the induction hypothesis is satisfied, since, in the acceptance game for $\mathcal{A}$, there are two non-closed environments, namely e_1 and its parent environment e^0. The priority of the fixpoint state associated to e_i is p, and the initial round is a p-round and the only non-closed round.

Now assume that the acceptance game for $\mathcal{A}^{\Sigma}_n$ over $T(\mathcal{A}, \mathbb{T})$ has proceeded to a configuration $C = (t, (O, \_), \_)$ that satisfies the induction hypothesis. Let r be the round of which C is the first configuration, and let $C' = (u, (\chi, e), \_)$ be the configuration in the game for $\mathcal{A}$ that induces t. Depending on the form of χ, $\mathcal{V}$ moves in the acceptance game of $\mathcal{A}^{\Sigma}_n$ as follows:

- If χ is P or $\overline{P}$ then the acceptance game of $\mathcal{A}$ ends. If the label of t is T then $\mathcal{V}$ wins both games, otherwise, the label is F and $\mathcal{S}$ wins both games.

- If $\chi = \chi_1 \vee \chi_2$, then t is labeled by N since C' is a disjunction configuration. Hence, r is an N-round and $\mathcal{V}$ picks a subtree in the game for $\mathcal{A}_n^\Sigma$. Finally, since r is an N-round, the condition on non-closed environments in the game for $\mathcal{A}$, respectively non-closed rounds in the game for $\mathcal{A}_n^\Sigma$ is satisfied: The former sequence does not change, and the latter is extended by r, which, by the behavior of N-rounds, is the parent of the next round.

- If $\chi = \chi_1 \wedge \chi_2$, then r is a U-round, and $\mathcal{S}$ decides whether the game for $\mathcal{A}_n^\Sigma$ continues in $t0$ or $t1$. Similarly to the case of disjunctions, the sequence of non-closed rounds is extended by r, a U-round, while the sequence of non-closed environments stays the same. As in the case of N-rounds, r is the parent of the next round.

- If $\chi = \langle a \rangle \chi'$, then as in the previous cases, r is an N-round and $\mathcal{V}$ can choose the correct successor in the game for $\mathcal{A}_n^\Sigma$. The conditions on non-closed environments and rounds is satisfied at the beginning of the next round as before.

- If $\chi = [a]\chi'$, the argument is completely symmetric to the previous case.

- If $\chi = \chi_1\, \chi_2$, then r is an N-round. $\mathcal{V}$ picks either successor of t. Since no new environment is created in the game for $\mathcal{A}$, and the game for $\mathcal{A}_n^\Sigma$ is extended by an N-round, the condition on non-closed environments and rounds is satisfied at the beginning of the next round.

- If $\chi = Q$, let i be the priority of Q. Note that r is an i-round, and that either successor of t is induced by the configuration $C'' = (u, (\delta(Q), e'), \varepsilon)$ where e' is a new and, hence not closed, environment with parent environment e. Hence, at the end of r, the sequence of non-closed environments is extended by e with an associated fixpoint configuration where the priority of the fixpoint is i. On the other hand, the sequence of non-closed rounds is extended by r, which is an i-round, and the parent of the next round. Hence, the condition regarding the sequences of non-closed environments and rounds is satisfied at the end of r.

- If $\chi = f$, then the next configuration in the game for $\mathcal{A}$ is $(u, (\mathsf{lookup}(f), e), \_)$, and e is closed. By assumption, it is winning for $\mathcal{V}$. Hence, since r is a V-round, r is closed at the end of itself. Moreover, if the sequence of non-closed rounds is $r_1, \dots, r_k, \dots, r_{k+1}, \dots, r_{k+m} = r$ where r_k is an i-round that is not inactive, for some i, but all the r_{k+j} for $1 \leq j \leq m$ are N-rounds, U-rounds, or inactive i'-rounds, then after the end of r, by Lemma 6.2.8, the rounds $r_{k+1}, \dots, r_{k+m}$ are closed, and r_k is inactive. Finally r_k is the parent of the next round. It follows that the condition on non-closed environments and rounds is satisfied at the beginning of the next round, since the former is reduced by e, and the latter is reduced by $r_{k+1}, \dots, r_{k+m}$ and, moreover, r_k is now inactive. Finally, r_k is an i-round such that the priority of the fixpoint state in the fixpoint configuration associated to e is i.

The play of $\mathcal{A}_n^\Sigma$ is winning for $\mathcal{V}$ if and only if the play for $\mathcal{A}$ is winning for her. By Lemma 6.2.4, a fixpoint configuration in that play is on the infinite path in the associated unfolding tree if and only if the associated environment is not closed. However, since the full induction invariant is maintained through the play of the game for $\mathcal{A}_n^\Sigma$, an environment in the play for $\mathcal{A}$ is never closed if and only if the corresponding i-round never becomes inactive or closed. Hence, the lower environment of the corresponding i-round is also never closed, whence the associated fixpoint configuration is on the infinite path of the unfolding tree of the play for $\mathcal{A}_n^\Sigma$. However, the environments of all other rounds, in particular the upper environments of inactive i-rounds, have priority 1. It follows that p is the highest priority of a fixpoint configuration that occurs infinitely often on the infinite path in the unfolding tree of the play for $\mathcal{A}$ if and only if p is the highest index of a fixpoint configuration, necessarily with fixpoint state Q_p, that occurs infinitely often on the infinite path in the unfolding tree of the play for $\mathcal{A}_n^\Sigma$. □

Theorem 6.2.10. *Let $\mathbb{T}$ be a tree with labels in $\mathcal{P}_n$. Let $\mathcal{A} \in \Sigma_1^n$ and $\mathcal{A}' \in \Pi_1^n$. Then $\mathbb{T}, \varepsilon \models \mathcal{A}$ if and only if $T(\mathcal{A}, \mathbb{T}), \varepsilon \models \mathcal{A}_n^\Sigma$, and $\mathbb{T}, \varepsilon \models \mathcal{A}'$ if and only if $T(\mathcal{A}', \mathbb{T}), \varepsilon \models \mathcal{A}_n^\Pi$.*

Proof. For the sake of notational simplicity, we only show that if $\mathcal{V}$ has a winning strategy for the acceptance game of $\mathcal{A}$ over $\mathbb{T}$, then she has a winning strategy for $\mathcal{A}_n^\Sigma$ over $T(\mathcal{A}, \mathbb{T})$. The proof for the respective result for $\mathcal{S}$ is symmetric, and the result for the case of Π^n is identical.

The proof is by induction over the length of the acceptance game of $\mathcal{A}_n^\Sigma$ over $T(\mathcal{A}, \mathbb{T})$, respectively over the number of its rounds. Assume that $\mathcal{V}$ has a winning strategy for the acceptance of $\mathcal{A}$ over $\mathbb{T}$. Since, by Lemma 6.2.27, she wins any play of the game for $\mathcal{A}_n^\Sigma$ if this play is winning for her in the induced play for $\mathcal{A}$ in the sense of that lemma, it is enough for $\mathcal{V}$ to make sure that she maintains the play for $\mathcal{A}_n^\Sigma$ in configurations that induce such a winning play. Surely, this is satisfied for the starting round, since it is played on $\varepsilon \in T(\mathcal{A}, \mathbb{T})$ which is induced by the initial configuration of the game for $\mathcal{A}$ over $\mathbb{T}$. This configuration is winning for $\mathcal{V}$ by assumption.

Now assume that $\mathcal{V}$ has played the game for $\mathcal{A}_n^\Sigma$ such that the induced game for $\mathcal{A}$ follows her winning strategy, and assume that the play for $\mathcal{A}_n^\Sigma$ is at the beginning of a round r. We show that she can play this round such that the induced play continues to be played according to her winning strategy. Let t be the node that r is played on and let $C' = (u, (\chi, e), \_)$ be the configuration that induces t.

- If χ is P or $\overline{P}$, then r is a T-round and $\mathcal{V}$ wins by assumption.
- If $\chi = \chi_1 \vee \chi_2$, let χ_i be the substate dictated by $\mathcal{V}$'s winning strategy for the game for $\mathcal{A}$. Since r is an N-round, $\mathcal{V}$ can choose the next node in the game for $\mathcal{A}_n^\Sigma$ and force the next round to be played on the node induced by $(u, (\chi_i, e), \_)$. Hence, the induced play for $\mathcal{A}$ continues to be played according to $\mathcal{V}$'s winning strategy.
- If $\chi = \chi_1 \wedge \chi_2$, by assumption $\mathcal{V}$ has a winning strategy for either successor configuration in the game for $\mathcal{A}$. Hence, no matter which node $\mathcal{S}$ chooses in the U-round r, the induced play for $\mathcal{A}$ continues to be played according to $\mathcal{V}$'s winning strategy.
- If $\chi = \langle a \rangle \chi'$ or $\chi = [a]\chi'$, by reasoning similar to that for the boolean operators, $\mathcal{V}$ can either force the induced play for $\mathcal{A}$ to be played according to her winning strategy, or it will be played according to her strategy no matter which successor $\mathcal{S}$ chooses.

- If $\chi = f$ or $\chi = \chi_1\,\chi_2$ or $\chi = Q$, both moves are deterministic, whence the induced play for $\mathcal{A}$ continues to be played according to $\mathcal{V}$'s winning strategy.

Hence, $\mathcal{V}$ can play the game for $\mathcal{A}_n^\Sigma$ such that the induced play for $\mathcal{A}$ is winning for her. By Lemma 6.2.27, she then also wins the play of the game for $\mathcal{A}_n^\Sigma$. $\square$

The Strictness Result

Theorem 6.2.11. *The alternation hierarchy is strict for the class of order-*1 APKA *over the class of fully infinite binary trees with labels in* $\mathcal{P}_n$.

Proof. We show that, for all $n \geq 1$, the automaton $\mathcal{A}_n^\Sigma$ is not in Π_1^n. Hence, $\Sigma_1^n \neq \Pi_1^n$ and $\Sigma_1^n \subsetneq \Sigma_1^{n+1}$. The proof for $\mathcal{A}_n^\Pi$ and Π_1^n is symmetric.

Recall from Example 2.1.10 that the space of fully infinite binary trees with labels in $\mathcal{P}_n$ is a complete metric space, where $d(\mathbb{T}_1, \mathbb{T}_2)$ is defined as 2^{-i} if i is the first level at which $\mathbb{T}_1$ and $\mathbb{T}_2$ do not agree, and 0 else. Moreover, for any APKA $\mathcal{A}$, the mapping $f_\mathcal{A}\colon \mathbb{T} \mapsto T(\mathbb{T}, \mathcal{A})$ is a contraction: If $\mathbb{T}_1$ and $\mathbb{T}_2$ differ at level i, but agree on levels $0, \dots, i-1$, then this manifests itself in the associated acceptance games after $i+1$ many moves at the earliest, since i modal diamonds are required to reach the difference, and the transition moving from $(\varepsilon, (Q_I, e^0), \varepsilon)$, where Q_I is the initial state of $\mathcal{A}$, also cannot be avoided. By the Banach Fixpoint Theorem (Theorem 2.1.11), the mapping $f_\mathcal{A}$ has a fixpoint $T^*_\mathcal{A}$, which is the limit of the sequence generated by starting with any tree $\mathbb{T}$ and repeatedly applying $f_\mathcal{A}$. By the definition of $T^*_\mathcal{A}$ we have $T(\mathcal{A}, T^*_\mathcal{A}) = T^*_\mathcal{A}$.

Now assume that $\mathcal{A}_n^\Sigma \in \Pi_1^n$. Then, by Observation 6.1.2, we have that $\overline{\mathcal{A}_n^\Sigma} \in \Sigma_1^n$, whence there is $\mathcal{A}' \equiv \overline{\mathcal{A}_n^\Sigma}$ with priorities actually in $1, \dots, n$, respectively $0, \dots, n-1$. Let $T^*_\mathcal{A}$ be the fixpoint of $f_{\mathcal{A}'}$. Then $T^*_\mathcal{A} = T(\mathcal{A}', T^*_\mathcal{A})$ and, hence $T^*_\mathcal{A}, \varepsilon \models \mathcal{A}_n^\Sigma$ if and only if $T^*_\mathcal{A}, \varepsilon \models \mathcal{A}' = \overline{\mathcal{A}_n^\Sigma}$, which is a contradiction. Hence, $\mathcal{A}_n^\Sigma \notin \Pi_1^n$. $\square$

Corollary 6.2.12. *The alternation hierarchy is strict for the class of order-*1-APKA *over the class of all structures.*

Since no APKA in Π_1^n is equivalent to $\mathcal{A}_n^\Sigma$ over the class of fully infinite binary trees, no APKA in Π_1^n can be equivalent to $\mathcal{A}_n^\Sigma$ over the class of all LTS.

Limits of the Approach

The approach used in this section does not generalize to APKA of order 2 or greater. The argument from Lemma 6.2.4 can be extended to a sufficient condition for infinite plays. This means that an occurrence of a fixpoint configuration associated to an environment e does not label a node on the infinite path of the unfolding tree if a ground type variable configuration $(\_, (x, e), \varepsilon)$ appears in the game. However, the occurrence of such a configuration is not necessary for a fixpoint configuration to label a node not on the infinite path.

Example 6.2.13. Consider the APKA $\mathcal{A}$ defined via

$$\begin{aligned} I\colon () &\mapsto X\,S \\ S\colon (x\colon \bullet) &\mapsto S\,x \vee X\,S \\ X\colon (f\colon \bullet \to \bullet) &\mapsto f\,P \end{aligned}$$

where the priorities of the individual fixpoint states are not important. A typical play of the acceptance game of $\mathcal{A}$ over some LTS is

$$\begin{aligned}
&(\_\,, (I, e^0), \varepsilon) && \\
&(\_\,, (X\,S, e_1), \varepsilon) && e_1 = (\emptyset, 1) \\
&(\_\,, (X, e_1), (S, e_1)) && \\
&(\_\,, (f\,P, e_2), \varepsilon) && e_2 = (f \mapsto (S, e_1), 2) \\
&(\_\,, (f, e_2), (P, e_2) && \\
&(\_\,, (S, e_1), (P, e_2) && \\
&(\_\,, (S\,x \vee X\,S, e_3), \varepsilon) && e_3 = (x \mapsto (P, e_2), 3) \\
&(\_\,, (S\,x, e_3), \varepsilon) && \\
&(\_\,, (S, e_3), (x, e_3)) && \\
&(\_\,, (S\,x \vee X\,S, e_4), \varepsilon) && e_4 = (x \mapsto (x, e_3), 4) \\
&(\_\,, (X\,S, e_4), \varepsilon) && \\
&(\_\,, (X, e_4), (S, e_4)) && \\
&(\_\,, (f\,P, e_5), \varepsilon) && e_5 = (f \mapsto (S, e_4), 5) \\
&(\_\,, (f, e_5), (P, e_5)) && \\
&(\_\,, (S, e_4), \varepsilon) && \\
&\ldots &&
\end{aligned}$$

and generates the left unfolding tree shown in Figure 6.1. Clearly the occurences of X are not on the infinite path, yet since X has no arguments of ground type, no ground type variable configurations in the associated environment occur in the play.

Of course, X does not occur in the defining formula of itself and, hence, can easily be seen to not be recursive in the previous example. However, consider a refined version of this example.

Example 6.2.14. Let the APKA $\mathcal{A}'$ be defined via

$$\begin{aligned}
I &\colon () \mapsto (X\,S) \\
S &\colon (x\colon \bullet) \mapsto x \vee S\,x \vee X\,S \\
X &\colon (f\colon \bullet \to \bullet) \mapsto (f\,(X\,f))\,P
\end{aligned}$$

where, again, the priorities of the individual fixpoint states are not important.

A typical run of $\mathcal{A}'$ is

$$\begin{aligned}
&(\_\,, (I, e^0), \varepsilon) && \\
&(\_\,, (X\,S, e_1), \varepsilon) && e_1 = (\emptyset, 1) \\
&(\_\,, (X, e_1), (S, e_1)) && \\
&(\_\,, (f\,(X\,f), e_2), \varepsilon) && e_2 = (f \mapsto (S, e_1), 2) \\
&(\_\,, (f, e_2), (X\,f, e_2) && \\
&(\_\,, (S, e_1), (X\,f, e_2) && \\
&(\_\,, (x \vee S\,x \vee X\,S, e_3), \varepsilon) && e_3 = (x \mapsto (X\,f, e_2), 3) \\
&(\_\,, (S\,x, e_3), \varepsilon) &&
\end{aligned}$$

$$\begin{array}{rl}
(\_\,, (S, e_3), (x, e_3)) & \\
(\_\,, (x \vee S\,x \vee X\,S, e_4), \varepsilon) & \quad e_4 = (x \mapsto (x, e_3), 4) \\
(\_\,, (X\,S, e_4), \varepsilon) & \\
(\_\,, (X, e_4), (S, e_4)) & \\
(\_\,, (f\,(X\,f), e_5), \varepsilon) & \quad e_5 = (f \mapsto (S, e_4), 5) \\
(\_\,, (f, e_5), (X\,f, e_5)) & \\
(\_\,, (S, e_4), \varepsilon) & \\
\dots &
\end{array}$$

which generates the right unfolding tree in Figure 6.1. In this case, X actually appears in its argument and, hence, is at least theoretically recursive, but, if the play continues like this, i.e., with $\mathcal{V}$ always chosing the right or middle disjunct in the transition relation of S, then the occurrences of X are not on the infinite path. On the other hand, if, at some point, $\mathcal{V}$ decides to choose the left disjunct in the transition relation of S, then the occurrence of $X\,f$ bound to x in the environment in question is potentially on the infinite path.

Remark 6.2.15. Example 6.2.13 in conjunction with Example 6.2.14 shows that Arnold's approach does not generalize beyond order 1, since it is not possible to find a condition that, already during a play of the acceptance game, reliably decides whether an occurrence of a fixpoint configuration is on the infinite path and, hence, relevant towards the acceptance condition. This does not mean that the alternation hierarchy necessarily collapses for HFL beyond order 2, nor that Arnold's approach cannot be mimicked in another automaton model for HFL. The latter, however, appears unlikely, given that the infinite path of an unfolding tree very naturally isolates infinite recursion.

6.2.2 Strictness for Simple APKA

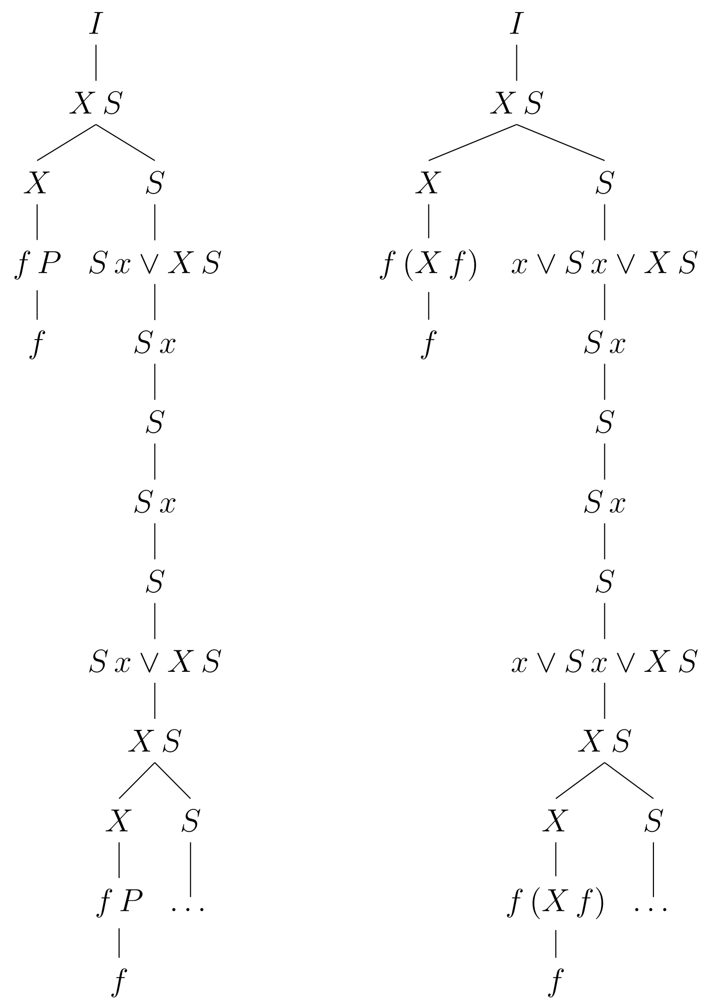
We now investigate the behavior so-called *simple* APKA. Simple APKA are defined by the analogue to the restriction in tail-recursive formulas that free fixpoint variables may not appear in operand position at applications. We then show that, for simple APKA, the acceptance condition degenerates to an ordinary parity condition, which also allows us to show strictness of the alternation hierarchy for simple APKA. We close the section with a discussion of the relationship between tail-recursive formulas and simple APKA.

Simple APKA

Definition 6.2.16. An APKA $\mathcal{A}$ with fixpoint state set $\mathcal{Q}$ is called *simple* if there is a partition $\mathcal{Q}_1, \dots, \mathcal{Q}_k$ of $\mathcal{Q}$ such that, for each $Q \in \mathcal{Q}_i$, the transition relation $\delta(Q)$ can be derived from the symbol χ_i in the grammar in Figure 6.2 where $Q'_i \in \mathcal{Q}_i$ and $f \in \{f_1^Q, \dots, f_{k_Q}^Q\}$ and $P \in \mathcal{P}$ and $a \in A$.

The essence of the definition is that an APKA is simple if fixpoint states appear in operand position in the transition relation of another fixpoint only if they belong to a partition set with lower index. In particular, operands in the transition relations of fixpoint states in $\mathcal{Q}_1$ can not contain fixpoint states themselves. This parallels the

Figure 6.1: The unfolding trees of the runs from Example 6.2.13 and Example 6.2.14.



decreasing recursion depth for operands in tail-recursive formulas. However, simple APKA are not restricted in the way boolean and modal operators are used. For example, a fixpoint definition of the form

$$Q\colon () \mapsto_i \langle a\rangle Q \wedge [a]Q$$

is permitted in a simple APKA, but is obviously problematic in the context of tail recursion. Note that order-0 APKA, i.e., PA, are always simple.

Observation 6.2.17. If $\mathcal{A}$ is simple, then $\overline{\mathcal{A}}$ is also simple.

The grammar in Definition 6.2.16 induces a labeling that associates a number in $\{0, \ldots, k\}$ to each substate in the transition relation of each fixpoint state. This labeling also works for $\overline{\mathcal{A}}$ and yields a valid partitioning that qualifies $\overline{\mathcal{A}}$ to be simple.

Example 6.2.18. The APKA $\mathcal{A}'$ given by the definitions

$$\begin{aligned}
I\colon () &\mapsto_0 F\,S \\
F\colon (f\colon \bullet \to \bullet, x\colon \bullet) &\mapsto_1 (f\,x) \vee (F\,(D\,f))\,x \\
D\colon (g\colon \bullet \to \bullet, y\colon \bullet) &\mapsto_0 g\,(g\,y) \\
S\colon (z\colon \bullet) &\mapsto_0 \langle a\rangle z
\end{aligned}$$

Figure 6.2: Grammar for the transition relation of simple APKA.

$$\begin{aligned}
\chi_i &::= P \mid \overline{P} \mid \chi_i \vee \chi_i \mid \chi_i \wedge \chi_i \mid \langle a\rangle\chi_i \mid [a]\chi_i \mid \chi_i\,\chi_{i-1} \mid Q'_i \mid f \mid \chi_{i-1} \\
\chi_{i-1}, \zeta_{i-1} &::= P \mid \overline{P} \mid \chi_{i-1} \vee \chi_{i-1} \mid \chi_{i-1} \wedge \chi_{i-1} \mid \langle a\rangle\chi_{i-1} \mid [a]\chi_{i-1} \\
&\qquad\qquad\qquad\qquad \mid \chi_{i-1}\,\chi_{i-2} \mid Q'_{i-1} \mid f \mid \chi_{i-2} \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
\chi_1, \zeta_1 &::= P \mid \overline{P} \mid \chi_1 \vee \chi_1 \mid \chi_1 \wedge \chi_1 \mid \langle a\rangle\chi_1 \mid [a]\chi_1 \mid \chi_1\,\zeta_0 \mid Q'_1 \mid f \mid \chi_0 \\
\chi_0, \zeta_0 &::= P \mid \overline{P} \mid \chi_0 \vee \chi_0 \mid \chi_0 \wedge \chi_0 \mid \langle a\rangle\chi_0 \mid [a]\chi_0 \mid \chi_0\,\zeta_0 \mid f
\end{aligned}$$

is simple by assigning partition level 1 to D and S, partition level 2 to F, and partition level 3 to I. The APKA $\mathcal{A}$ from Example 4.1.2, given by the definitions

$$\begin{aligned}
I\colon () &\mapsto_0 H\,S \\
H\colon (h\colon \bullet \to \bullet) &\mapsto_0 (F\,h)\,(P \wedge H\,h) \\
F\colon (f\colon \bullet \to \bullet, x\colon \bullet) &\mapsto_1 (f\,x) \vee (F\,(D\,f))\,x \\
D\colon (g\colon \bullet \to \bullet, y\colon \bullet) &\mapsto_0 g\,(g\,y) \\
S\colon (z\colon \bullet) &\mapsto_0 \langle a\rangle z
\end{aligned}$$

is not simple since H appears on the operand side of an application in its own transition relation.

Lemma 6.2.19. *Let $\mathcal{A}$ be a simple* APKA *with fixpoint states partitioned into classes $\mathcal{Q}_1, \dots, \mathcal{Q}_k$. Let $(C_i)_{i\in\mathbb{N}}$ be an infinite play of the acceptance game over some pointed LTS. Then the infinite path in the unfolding tree of the play always continues through the leftmost son of a branching node except at most $k-1$ times. In particular, there is $n \in \mathbb{N}$ such that, starting with the configuration C_n, we have that C_{i+1} labels the unique son of the node that labels C_i.*

Proof. Let $\mathbb{T}$ be the unfolding tree of the play. We can extend the labeling of the substates of the transition relation by numbers in $\{0, \dots, k\}$ to the configurations in the run by labeling a configuration by the number associated to the substate of its closure component. Note that, on all paths in $\mathbb{T}$, the labeling on the sequence of configurations on the path is non-increasing. This follows by a simple induction from the fact that lambda variable configurations only occur on leaves in the unfolding tree: Let $C = (\_, (\chi, \_), \_)$ be a non-leaf configuration in the unfolding tree which labels t, and let it be labeled by k', i.e., k' is the smallest natural number i such that χ was derived from χ_i in the grammar in Definition 6.2.16. If χ is of the form $\chi_1 \vee \chi_2$, $\chi_1 \wedge \chi_2$, $\langle a\rangle\chi_1$ or $[a]\chi_1$, the substate of the next configuration, which labels t's unique successor in $\mathbb{T}$, is χ_1 or, in one of the first two cases, possibly χ_2. From the grammar it is easy to see that the label of this substate is at most k'. If $\chi = \chi_1\,\chi_2$, the leftmost son of t is labeled by $(\_, (\chi_1, \_), \_)$, and any non-leftmost son is labeled by $(\_, (\chi_2, \_), \_)$. In the first case, by the same reasoning as before, the labeling for the next configuration is non-increasing, and in the second case, it is actually strictly decreasing to at most $(k'-1)$. Finally, if $\chi = Q \in \mathcal{Q}_{k'}$ then the unique son of t is labeled by $(\_, (\delta(Q), \_), \_)$ which is labeled by at most k' by definition.

In fact, from the above we can see that, on any path, the labeling of the configurations strictly decreases whenever the path takes a non-leftmost son, unless the labeling has already reached 0. However, this does not happen on the infinite path, since substates derived from χ_0 contain no fixpoint variables. Hence, from a configuration with a substate derived from χ_0, the play either ends outright after finitely many steps, or a variable configuration is reached. Since these do not occur on the infinite path, no configuration labeled by 0 appears on it. It follows that the infinite path continues through a non-leftmost son of a branching node at most $k-1$ times.

The second statement follows from the first: Let C_n be the last configuration that labels a node on the infinite path that is a non-leftmost son, or the initial configuration if no such configuration exists. Since the son configuration of C_n and each subsequent configuration is also on the infinite path, no variable configurations with index greater than or equal to n appear in the play, and the unfolding tree degenerates to a simple path as in the lemma. $\square$

Remark 6.2.20. This restriction on the behavior of the infinite path of the unfolding tree of a run, and, hence, the infinite recursion in simple APKA, is quite drastic. We have seen in Chapter 4, that it is generally highly nontrivial to extract the proper contents of infinite recursion out of effects introduced by higher-order constructs. By restricting the interplay between fixpoint states and function application, all this collapses to a simple parity condition. In particular, this collapse is far more remarkable than the LIFO-properties we identified for order-1 APKA, given that simple APKA can have any type-theoretic order. This suggests that the restrictions of simple APKA, respectively tail-recursive formulas (see Section 6.2.2 for a discussion of the relation between tail recursion and simple APKA) both fundamentally cut off the interplay between recursion and higher-order.

Corollary 6.2.21. *Let $\mathcal{A}$ be a simple* APKA *with fixpoint states partitioned into classes $\mathcal{Q}_1, \dots, \mathcal{Q}_k$. Let $(C_i)_{i \in \mathbb{N}}$ be an infinite play of the acceptance game over some pointed LTS. Let Q be a fixpoint state of $\mathcal{A}$. Then Q occurs infinitely often in fixpoint configurations during the play if and only if it occurs infinitely often in fixpoint configurations on the infinite path of the unfolding tree of the play.*

Proof. Since, by Lemma 6.2.19 the infinite path comprises all configurations from some point on, only finitely many configurations are not on the infinite path, in particular, only finitely many fixpoint configurations. $\square$

Corollary 6.2.21 tells us that, for the case of simple APKA, a standard parity condition is enough. This allows us to adopt Arnold's proof [4] to the case of simple APKA, since a parity condition is easily expressible for such an automaton.

We now define restrictions of the alternation classes to simple APKA.

Definition 6.2.22. For all $n \geq 1$, the class Σ^n_s is the class of all APKA that are equivalent to a simple APKA in Σ^n, and the class Π^n_s is the class of all APKA that are equivalent to a simple APKA in Π^n.

Remark 6.2.23. The relations between the classes observed in Observation 6.1.2 hold accordingly.

Note that the classes are defined with respect to *all* APKA, regardless of their order and actually being simple themselves. In the following, we will make clear when we mean that an APKA is actually simple, or explicitly pass to an equivalent simple APKA.

Encoding Runs into Trees

Similar to the approach for order-1 APKA, for each $n \geq 1$, define a set of propositions $\mathcal{P}_n^s$ as $\mathcal{P}_n^s = \{N, U, T, F, P_1, \dots, P_n\}$. Note that the proposition V is not needed in this case and, hence is missing from $\mathcal{P}_n^s$, which distinguishes $\mathcal{P}_n^s$ from $\mathcal{P}_n$.

Again, consider the class of fully infinite binary trees with labels in $\mathcal{P}_n^s$ and a single transition a. Given such a tree $\mathbb{T}$ and a simple APKA $\mathcal{A}$ of any order, but in Σ_s^n or Π_s^n, we encode the game graph of the acceptance game of $\mathcal{A}$ from the root of the tree into another fully infinite binary tree $T(\mathcal{A}, \mathbb{T})$. We inductively describe which configurations induce a node, and how these nodes are labeled. The intuition is the same as in the case for order-1-APKA, except that we ignore variable configurations.

The root of $T(\mathcal{A}, \mathbb{T})$ is induced by the initial configuration of the acceptance game of $\mathcal{A}$ over $\mathbb{T}$, i.e., $(\varepsilon, (Q_I, e^0), \varepsilon)$. The labeling of a node induced by a configuration $C = (t, (\chi, e), \Gamma)$ depends on the type of χ:

- If $\chi = P$ then the node induced by C is labeled by T if $\mathbb{T}, t \models P$ and it is labeled by F if $\mathbb{T}, t \not\models P$. Conversely, if $\chi = \overline{P}$, then the node induced by C is labeled by T if $\mathbb{T}, t \not\models P$ and by F if $\mathbb{T}, t \models P$. Both the left and the right successor of the node are induced by the successor configuration of C again.

- If $\chi = \chi_1 \vee \chi_2$ or $\chi = \chi_1 \wedge \chi_2$, the node is labeled by N, respectively by U. The left successor of the node is induced by the configuration $(t, (\chi_1, e), \Gamma)$ and the right successor of the node is induced by the configuration $(t, (\chi_2, e), \Gamma)$.

- If $\chi = \langle a \rangle \chi'$ or $\chi = [a]\chi$, then the node is labeled by N, respectively by U. The left successor of the node is induced by the configuration $(t0, (\chi', e), \Gamma)$, while the right successor of the node is induced by the configuration $(t1, (\chi', e), \Gamma)$.

- if $\chi = \chi_1\, \chi_2$ or f, then the node is labeled by N and both its left and the right successor are induced by the successor configuration of C.

- If $\chi = Q$, then the node is labeled by P_i, where $i = \Delta(Q)$ if $\mathcal{A} \in \Sigma^n$ and n is even, or if $\mathcal{A} \in \Pi^n$ and n is odd, respectively $i = \Delta(Q) + 1$ if $\mathcal{A} \in \Pi^n$ and n is odd, or if $\mathcal{A} \in \Sigma^n$ and n is even. In either case, both its left and the right successor are induced by the successor configuration of C.

Again, we make the tree binary and infinite by duplicating subtrees if necessary and adding repeating configurations at nodes where the game ends. Also, the root of $T(\mathcal{A}, \mathbb{T})$ is again labeled by P_i, where i, respectively $i - 1$, is the priority of the initial state of $\mathcal{A}$.

Automata That are Hard for Alternation Classes

Definition 6.2.24. Let $\mathcal{A}_n^\Sigma$ and $\mathcal{A}_n^\Pi$ be defined as follows:

$$\begin{aligned} Q_1 &\colon () \mapsto_{\Delta(1)} O \\ &\vdots \qquad \vdots \\ Q_n &\colon () \mapsto_{\Delta(n)} O \\ O &\colon () \mapsto_{\Delta(1)} \chi \end{aligned}$$

where

$$\chi = T \vee \Big(\overline{F} \wedge \big(\overline{N} \vee \langle a\rangle O\big) \wedge \big(\overline{U} \vee [a]O\big) \wedge \big(\overline{P_1} \vee \langle a\rangle Q_1\big) \wedge \cdots \wedge \big(\overline{P_n} \vee \langle a\rangle Q_n\big)\Big)$$

and $\Delta(Q_i) = i$ for $\mathcal{A}_n^\Sigma$ if n is even and for $\mathcal{A}_n^\Pi$ if n is odd, respectively $\Delta(Q_i) = i - 1$ for $\mathcal{A}_n^\Sigma$ if n is odd and for $\mathcal{A}_n^\Pi$ if n is even. Moreover, $\Delta(O) = \Delta(Q_1)$. The initial state is O in either case.

Clearly, $\mathcal{A}_n^\Sigma$ is in Σ^n and $\mathcal{A}_n^\Pi$ is in Π^n. Since both automata are of order 0, they are simple. Note that these automata are direct derivatives of Walukiewicz' formulas encoding the winner of a parity game (see Example 2.2.4).

Note that a run of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ is much simpler than one of the automata from Section 6.2.1. Starting from an occurrence of a fixpoint configuration in 0, a play of the acceptance game for either automaton ends after at most four moves, or the automaton makes a transition and either ends up back in a fixpoint configuration in O or ends up in such a configuration after passing through one of the Q_i. Hence, we can use the notion of a *round* again.

Definition 6.2.25. Let $(C_i)_{i \in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over a tree $T(\mathcal{A}, \mathbb{T})$ where $\mathcal{A}$ is simple and $\mathcal{A} \in \Sigma^n$, respectively $\mathcal{A} \in \Pi^n$. A *round* is a sequence of configurations, starting in a configuration of the form $(t, (O, \_), \_)$ and contains all configurations up to, but not including the next configuration of this form, if it exists. A round is *played on* the node $t \in T(\mathcal{A}, \mathbb{T})$ if t is the vertex component of the first configuration of the round.

A round is called an F-round, T-round, N-round, U-round, or an i-round if the vertex it is played on is such that $\mathcal{L}(t)$ is the singleton containing F, T, N, U or P_i, respectively.

Note that for a round played on some node t, not all configurations of the round have vertex component t.

In contrast to the notion of a round for the case of order-1 APKA, we do not distinguish the initial round, nor do we deal with environments associated to rounds. Since for simple APKA, the sequence of fixpoint configurations that occur in a run is enough to determine acceptance, it is enough to track the sequence of rounds in a run of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ to determine acceptance of the automaton in question.

Note that in an F-round and a T-round, the play will end since one of the players has a winning strategy, and in an X-round, for $X = N, U, P_1, \dots, P_n$, it is not hard to see that $\mathcal{S}$ will force the play to go into the conjunct of the transition relation of O that starts with $\overline{X} \vee \dots$. Without loss of generality, we assume from now on that he does so.

Lemma 6.2.26. *Let $(C_i)_{i \in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over a tree $T(\mathcal{A}, \mathbb{T})$, where $\mathbb{T}$ has labels in $\mathcal{P}_n^s$ and $\mathcal{A}$ is simple and $\mathcal{A} \in \Sigma^n$, respectively $\mathcal{A} \in \Pi^n$. The following hold for all rounds:*

- *During each round there is exactly one transition, unless the round is a T-round or an F-round. In this case, no transition occurs and the play ends after at most three moves. The target of the transition in this round is chosen by $\mathcal{S}$ in U-rounds and by $\mathcal{V}$ in N-rounds and i-rounds, for $1 \leq i \leq n$.*

- *In an i-round, for $1 \leq i \leq n$, fixpoint configurations in Q_i and O occur, while in all other rounds, only a single fixpoint configuration in O occurs.*

- *Every configuration in the play is part of exactly one round.*

Proof. By simple verification. □

Lemma 6.2.27. *Let $(C_i)_{i \in I}$ be a finite or infinite play of the acceptance game of $\mathcal{A}_n^\Sigma$ or $\mathcal{A}_n^\Pi$ over a tree $T'(\mathcal{A}, \mathbb{T})$, where $\mathbb{T}$ has labels in $\mathcal{P}_n^s$ and $\mathcal{A}$ is simple and $\mathcal{A} \in \Sigma^n$, respectively $\mathcal{A} \in \Pi^n$. Let $(r_i)_{i \in I'}$ be the sequence of rounds in that play, and let $(t_i)_{i \in I'}$ be the vertices these rounds are played on. Let $(C'_i)_{i \in I'}$ be the play of $\mathcal{A}$ over $\mathbb{T}$ such that C'_i induces t_i in $T(\mathcal{A}, \mathbb{T})$. Then $\mathcal{V}$ wins the play $(C'_i)_{i \in I'}$ if and only if she wins the play $(C_i)_{i \in I}$.*

Proof. It is immediate that the lemma holds for finite plays. For infinite plays, the proof is by induction on the length of the play of $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$. Note that the play $(C'_i)_{i \in I'}$ of $\mathcal{A}$ over $\mathbb{T}$ necessarily contains infinitely many fixpoint configurations. Moreover, by the definition of this play, there is a bijection between the rounds in the play of $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$ on one side, and the configurations in $(C'_i)_{i \in I'}$. Let p be the highest priority such that there are infinitely many fixpoint configurations with a fixpoint of priority p in the play of $\mathcal{A}$. If $\mathcal{A} \in \Sigma^n$ and n is even, respectively if $\mathcal{A} \in \Pi^n$ and n is odd, there are also infinite many p-rounds in the play of $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$. Moreover, for all $p' > p$, there are at most finitely many p'-rounds in the latter play, since otherwise, there would be infinitely many fixpoint configurations with fixpoints of priority p' in the play of $\mathcal{A}$. If, on the other hand, $\mathcal{A} \in \Sigma^n$ and n is odd, respectively if $\mathcal{A} \in \Pi^n$ and n is even, there are also infinite many $p + 1$-rounds in the play of $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$, and only finitely many p'-rounds if $p' > p$. Finally, the priority of O is the lowest of any fixpoint state in $\mathcal{A}_n^\Sigma$ and $\mathcal{A}_n^\Pi$, the priority of p-rounds is p if $\mathcal{A} \in \Sigma^n$ and n is even, or if $\mathcal{A} \in \Pi^n$ and n is odd, and the priority of $p+1$-rounds is p if $\mathcal{A} \in \Sigma^n$ and n is odd, or $\mathcal{A} \in \Pi^n$ and n is even. Hence, the highest priority that occurs infinitely often in the acceptance game for $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$ is also p. It follows immediately that $\mathcal{V}$ wins the play for $\mathcal{A}_n^\Sigma$, respectively $\mathcal{A}_n^\Pi$ over $T(\mathcal{A}, \mathbb{T})$ if and only if she wins the play for $\mathcal{A}$ over $\mathbb{T}$. □

Lemma 6.2.28. *Let $\mathcal{A} \in \Sigma^n$ and $\mathcal{A}' \in \Pi^n$ be simple* APKA. *Let $\mathbb{T}$ be a tree with labels in $\mathcal{P}_n^s$. Then $\mathbb{T}, \varepsilon \models \mathcal{A}$ if and only if $T(\mathcal{A}, \mathbb{T}), \varepsilon \models \mathcal{A}_n^\Sigma$, and $\mathbb{T}, \varepsilon \models \mathcal{A}'$ if and only if $T(\mathbb{T}, \mathcal{A}'), \varepsilon \models \mathcal{A}_n^\Pi$.*

Proof. This is proved exactly in the same way as Thm 6.2.10. □

The Strictness Result

The proof of the strictness result follows the pattern from [4] in the same way as the proof of Thm 6.2.11. We briefly give the necessary arguments.

Theorem 6.2.29. *The alternation hierarchy is strict for the class of simple* APKA *over the class of fully infinite binary trees with labels in $\mathcal{P}_n^s$.*

Proof. We show that, for all $n \geq 1$, the automaton $\mathcal{A}_n^\Sigma$ is not in Π^n. Hence, $\Sigma_s^n \neq \Pi_s^n$ and $\Sigma_s^n \subsetneq \Sigma_s^{n+1}$. The proof for $\mathcal{A}_n^\Pi$ and Π_s^n is symmetric.

Again, by Example 2.1.10, the space of fully infinite binary trees with labels in $\mathcal{P}_n^s$ is a complete metric space, where $d(\mathbb{T}_1, \mathbb{T}_2)$ is defined as 2^{-i}, where i is the first level at which $\mathbb{T}_1$ and $\mathbb{T}_2$ do not agree, and 0 else. Also, for any APKA $\mathcal{A}$, the mapping $f_\mathcal{A} \colon \mathbb{T} \mapsto T(\mathbb{T}, \mathcal{A})$ is a contraction: If $\mathbb{T}_1$ and $\mathbb{T}_2$ differ at level i, but agree on levels $0, \dots, i-1$, then this manifests itself in the associated acceptance game at the earliest after $i+1$ many moves, since i modal diamonds are required to reach the difference, and the transition moving from $(\varepsilon, (Q_I, e^0), \varepsilon)$, where Q_I is the initial state of $\mathcal{A}$ also cannot be avoided. By the Banach Fixpoint Theorem (Theorem 2.1.11), the mapping $f_\mathcal{A}$ has a fixpoint $T^*_\mathcal{A}$, which can be obtained as the limit of the sequence obtained by starting with any tree $\mathbb{T}$ and repeatedly applying $f_\mathcal{A}$. By the definition of $T^*_\mathcal{A}$ we have $T(\mathcal{A}, T^*_\mathcal{A}) = T^*_\mathcal{A}$.

Now assume that $\mathcal{A}^\Sigma_n \in \Pi^n_s$. Then, by Observation 6.1.2, we have that $\mathcal{A} = \overline{\mathcal{A}^\Sigma_n} \in \Sigma^n_s$. Let $T^*_\mathcal{A}$ be the fixpoint of $f_\mathcal{A}$. Then $T^*_\mathcal{A} = T(\mathcal{A}, T^*_\mathcal{A})$ and, hence $T^*_\mathcal{A}, \varepsilon \models \mathcal{A}^\Sigma_n$ if and only if $T^*_\mathcal{A}, \varepsilon \models \mathcal{A} = \overline{\mathcal{A}^\Sigma_n}$, which is a contradiction. Hence, $\mathcal{A}^\Sigma_n \notin \Pi^n_s$. $\square$

Corollary 6.2.30. *The alternation hierarchy is strict for the class of simple* APKA *over the class of all structures.*

Since no APKA in Π^n_s is equivalent to $\mathcal{A}^\Sigma_n$ over the class of fully infinite binary trees, no APKA in Π^n_s is equivalent to $\mathcal{A}^\Sigma_n$ over the class of all LTS.

Simple APKA and Tail Recursion

We briefly discuss the relation between tail recursion and simple APKA. At a glance, both formalisms appear to be incomparable, even though simple APKA are meant to be the APKA-side equivalent of tail-recursive formulas. The reason for them being incomparable is that simple APKA may contain unrestricted use of boolean and modal operators, which we need in order to encode the semantics for $\mathcal{A}^\Sigma_n$, respectively $\mathcal{A}^\Pi_n$. On the other hand, tail-recursive formulas can have fixpoint binders of low order that appear in a non-tail-recursive fashion, provided that this behavior appears in a fixpoint-variable-closed subformula.

However, consider Remark 5.3.9, where we gave an informal definition of *strictly tail-recursive* formulas as those whose proof of tail-recursiveness does not require rule $(\mathsf{fp_F})$ and which, hence, do not require non-tail-recursive fixpoint definitions of lower order. Every strictly tail-recursive formula is actually equivalent to a simple APKA. The necessary classification of subformulas which descends at applications is available via the notion of recursion depth, which does descend at the operand of an application. In fact, recursion depth overshoots its target since it also descends at, e.g., negations, which is not required for simple APKA.

Conversely, it is certainly possible to adapt the restrictions of boolean and modal alternation that exist for tail-recursive formulas to simple APKA by a further restriction of their transition relation. This, of course, strictly lowers the expressive power of such automata, since, e.g., it is not possible to express the behavior of the acceptance game encoded into a tree in such an automaton. However, we believe that it is possible to express the behavior of the acceptance game of a similarly restricted automaton if said game is encoded into a tree, provided the recursion depths match. This yields the following conjecture.

Conjecture 6.2.31. *The alternation hierarchy collapses for the class of all tail-recursive* HFL *formulas of fixed recursion depth.*

6.3 Fixpoint Polarity Switching

We now study combinations of classes of LTS and fragments of HFL where we can replace a fixpoint definition of one polarity by a fixpoint definition of the opposing polarity. This is the converse of what we studied in Section 6.2. Instead of showing strictness of the alternation hierarchy, now we are interested in first steps towards collapse results for the alternation hierarchy over restricted classes of LTS and for restricted classes of formulas. For $\mathrm{HFL}^0 = \mathcal{L}_\mu$ it is know that the alternation hierarchy collapses over the class of finite LTS which have no infinite paths [69], and a number of classes of finite LTS involving transitive transition relations [29, 1, 28]. See the introduction of [41] for an overview.

We show that for $\mathcal{L}_\mu$ we can rewrite least-fixpoint definitions as greatest-fixpoint definitions in HFL^1, and vice versa. However, this does not yield a collapse result, since most of the translations do only work for one fixpoint definition. The construction follows the characterization of fixpoints in Kleene's Theorem (Theorem 2.1.7), according to which the greatest fixpoint of a monotone operator can be obtained by starting with the top element of the respective lattice and then applying the operator until stabilization occurs. On a lattice of finite height, this procedure stabilizes after finitely many iterations.

For the sake of simplicity, consider a setting with only one action a. Let $\varphi_X = \nu(X : \bullet).\, \psi_X$ be a greatest-fixpoint definition. Consider the formula

$$\varphi'_X = \Big(\mu(F_X : \bullet \to \bullet).\, \lambda(x : \bullet).\, \big(x \wedge [a]^*(x \to \psi_X[x/X])\big) \vee F_X\,(\psi_X[x/X])\Big)\,\mathtt{tt}$$

where $[a]^*\psi$ is to be understood as an abbreviation of $\nu Y.\, \psi \wedge [a]Y$. The right disjunct encodes the iteration part, while the left disjunct encodes the stabilization test. If we consider that if $x \equiv X^i$, then $\varphi[x/X] \equiv X^{i+1}$, the formula can be unfolded to

$$\bigvee_{i \in \mathbb{N}} \big(X^i \wedge [a]^*(X^i \to X^{i+1})\big)$$

which holds on a vertex in a given LTS if and only if this vertex is also in the set defined by φ_X: Such a vertex is in the greatest fixpoint defined in φ_X if and only if it is in one of the finite approximations of said fixpoint, and if this approximation is stable, i.e., if it agrees with the fixpoint itself. The latter condition is expressed by the implication after the box. An approximation of a fixpoint is stable if and only if all vertices in the candidate fixpoint are also in the next approximation. The restriction to reachable vertices (via $[a]^*$) is sufficient due to invariance under bisimulation - we can always restrict attention to an LTS in which all vertices are reachable.

The resulting formula still contains a greatest-fixpoint definition under the disguise of $[a]^*$. However, this definition is clearly alternation-free with the fixpoint definition of F_X. However, it is not alternation-free with fixpoint whose variables occur freely in ψ_X. Hence, this translation as is does not yield a collapse result from $\mathcal{L}_\mu$ into HFL^1, but we believe it can be modified to yield such a collapse result, motivating the following conjecture.

Conjecture 6.3.1. *The $\mathcal{L}_\mu$ alternation hierarchy collapses into the alternation-free fragment of* HFL^1*.*

6.3.1 Polarity Switching for Monadic HFL^1 and HFL^2

For the sake of formula readability, we restrict the discussion to LTS with only one action a and one proposition P. This is solely for the purpose of exposition, the necessary extensions are straightforward. Moreover, for the time being, we only consider the class of finite LTS. We write $\equiv_{\mathsf{fin}}$ to denote equivalence of HFL formulas restricted to this class.

The Idea

We first present the approach for order 1 informally. Note that we restrict ourselves to fixpoints with just one argument.

Let $\varphi_X = \nu(X : \tau).\, \psi_X$, where $\tau = \bullet \to \bullet$, be a greatest-fixpoint definition such that $\psi_X \in \mathrm{HFL}^1$. Define φ'_X as

$$\Big(\mu(F_X \colon (\tau) \to \bullet \to \bullet).\, \lambda(f_X : \tau).\, \lambda(x : \bullet).\\ \big((f_X\, x) \wedge [a]^*(\varphi^1_H\, \varphi^1_t)\big) \vee \big(F_X\, \psi_X[f_X/X]\big)\, x\Big)\, \lambda(z : \bullet).\, \mathtt{tt}$$

where

$$\varphi^1_t = \lambda(y : \bullet).\, (f_X\, y) \to (\psi_X[f_X/X]\, y)$$

and

$$\begin{aligned}\varphi^1_H = \nu(H \colon (\tau) \to \bullet).\, \lambda(t : \tau).\, (t\, P) \wedge (t\, \neg P)\\ \wedge(H\, \lambda(z : \bullet)\, t(\langle a\rangle z)) \wedge (H\, \lambda(z : \bullet)\, t([a]z))\\ \wedge(H\, \lambda(z_1 : \bullet)\, (H\lambda(z_2 : \bullet)\, t\, (z_1 \vee z_2)))\\ \wedge(H\, \lambda(z_1 : \bullet)\, (H\lambda(z_2 : \bullet)\, t\, (z_1 \wedge z_2))).\end{aligned}$$

Then, for all $\psi \in \mathrm{HFL}^1_1$, we have that $\varphi_X\, \psi \equiv_{\mathsf{fin}} \varphi'_X\, \psi$. This construction is to be understood as follows: The formula φ'_X encodes the same approach as in the previous section, i.e., constructing subsequent approximations of the fixpoint of X, starting from the top element of the respective lattice. In this case, the top element is $\lambda(z : \bullet).\, \mathtt{tt}$. In the fixpoint F_X, the current approximation is stored in f_X, and the argument at which the fixpoint is to be evaluated is stored in x. The right disjunct $F_X\, \psi_X[f_X/X]$ builds the next iteration by replacing x_φ with $\psi_X[f_X/X]$.

The left disjunct follows the same pattern as for the order-0 case as well. The clause $f_X\, x$ returns the value of the fixpoint at the given argument, while the clause $[a]^*(\varphi^1_H\, \varphi^1_t)$ verifies that a sufficiently stable approximation is used. Consider first the formula φ^1_t. Over a given LTS, it expresses that if a vertex is in the semantics of $f_X\, y$, i.e, in the set defined by the evaluation of the current approximation at argument y, then the vertex is also in the semantics of $\psi_X[f_X/X]\, y$, i.e., the set defined by the evaluation of the next approximation at argument y. If we could actually evaluate φ^1_t at every subset of the underlying set of the LTS, this would exactly encode stabilization of the fixpoint. The reason for this is that in this case, the given approximation and the one after it agree as functions on all arguments. However, over a given LTS, many sets are not HFL definable since HFL is bisimulation invariant and, hence cannot define sets that are not unions of bisimulation classes. Moreover, over general (finite) LTS, it is not possible to enumerate all sets.[4]

[4]It is possible to enumerate all subsets of an LTS if the LTS is ordered [63].

However, closer inspection of the behavior of fixpoint stabilization at a given argument shows that it is not necessary to enumerate all subsets of a given LTS. In fact, it is sufficient to test for stabilization in all sets defined by an HFL^1 formula. Clearly, this is a necessary condition since an approximation that does not agree with the subsequent approximation on some HFL^1-formula cannot be final. However, stabilization on all sets defined by an HFL^1 formula is also a sufficient condition. The intuitive reasoning behind this is that the argument in question is an HFL^1 formula, and each approximation, including the final one, is created from the previous one via manipulation with the operators available in HFL^1. This behavior allows us to prove that an approximation that is stable on all HFL^1 formulas at some point agrees with all subsequent approximations on HFL^1-definable arguments. This includes agreement with the final approximation. Hence, testing on the set of all HFL^1-definable sets is enough.

Given the presence of fixpoints and lambda abstraction in HFL^1 formulas, it appears equally hopeless to enumerate all HFL^1 formulas. However, over a given finite LTS, each HFL^1 formula is actually equivalent to one in Basic Modal Logic. The reason for this is twofold. First, over a fixed finite LTS, each formula is equivalent to one not containing fixpoint operators and, hence, no fixpoint variables, since each formula is equivalent to a finite unfolding. Second, each fixpoint-free formula in HFL^1 is equivalent, over a given finite structure, to one in Basic Modal Logic: Starting from a formula containing lambda abstraction and application, note that the formula, as a syntactic object, is in λ_{ML} (cf. Section 2.3). Moreover, both α-conversion, i.e., renaming of variables and their binders, as well as β-reduction maintain semantics of HFL formulas. Using strong normalization of the Simply-Typed Lambda Calculus, (cf. Theorem 2.3.1), we obtain a semantically equivalent formula that contains no redexes. Since this formula, again, is well-typed, it cannot contain lambda abstraction, since the formula is of ground type and, for an order-1 abstraction, any argument would yield a valid redex, contradicting that the formula is in β-normal-form. This also precludes presence of order-1 subterms of the formula, whence it must be in ML.

The formula φ^1_H now codes the second part of the stabilization test by iterating over all ML-definable sets. It consumes an argument t of type $\bullet \to \bullet$, which initially is the test encoded in φ^1_t, which itself consumes a ground type argument and returns whether the approximation of the fixpoint of X stored in f_X is stable on this argument. φ^1_H, after unfolding, is the infinite conjunction $\bigwedge_{\psi \in \mathsf{ML}} t\,\psi$. It is not hard to see that this is at least true for atomic formulas in ML. For more complex formulas, consider, for example, the subformula

$$H\,(\lambda(z\colon \bullet).\, t(\langle a\rangle z)).$$

It constitutes a recursive call to H with a new function argument. This function consumes a ground type object stored in z and returns $t\,\langle a\rangle z$, i.e., applies the stabilization test in t to the formula $\langle a\rangle z$, where z stands for an as of yet undefined subformula. Hence, in the recursive call, this undefined subformula is, for example, filled with P via the third conjunct, which applies t to P. Since, in this iteration of the fixpoint, t is the function $z \mapsto t\,\langle a\rangle z$ (where the t in the formula now contains the original test, i.e., φ^1_t), this is equivalent to $\varphi^1_t\langle a\rangle P$. Continuing with this principle, the equality above can be seen to hold.

Putting it all together, let $\psi \in \mathrm{HFL}^1$. We argue in terms of the HFL model-checking game that $\varphi_X\, \psi \equiv_{\mathsf{fin}} \varphi'_X\, \psi$. Let $\mathcal{T}$ be an LTS and let v be a vertex in $\mathcal{T}$. If $v \in [\![\varphi_X\, \psi]\!]^\eta_{\mathcal{T}}$ for some interpretation η, then also $v \in [\![\varphi'_X\, \psi]\!]^\eta_{\mathcal{T}}$. Let n be the least number such that the nth approximation of X is equivalent to the fixpoint itself over $\mathcal{T}$. Then $\mathcal{V}$, in the model-checking game, chooses the right disjunct in φ'_X for n times. After n such iterations, the variable f_X contains the nth approximation of the fixpoint of X and, hence, is equivalent to it. If $\mathcal{V}$ now chooses the left disjunct, the formula $f_X\, x$ evaluates to true since x contains the semantics of ψ. On the other hand, since f_X contains an approximation that agrees with the true value of the fixpoint, this approximation is stable on all arguments. In particular, it is stable on all HFL^1-definable arguments.

Conversely, assume that $v \notin [\![\varphi_X\, \psi]\!]^\eta_{\mathcal{T}}$. Now, let n be the least number such that the nth approximation of X is stable on all HFL^1-definable arguments. In the model-checking game for $\varphi'_X\, \psi$, there are two possibilities for $\mathcal{V}$. She can always choose the right disjunct, or she can choose it for m iterations of F_X and then choose the left disjunct. In the first case, $\mathcal{V}$ eventually loses since the associated signature for F_X reaches 0. In the second case, there are two subcases. If $m \geq n$, then $f_X\, x$ evaluates to false, since f_X contains the mth approximation of the fixpoint of X and, hence the true value of it on arguments definable in HFL^1. Since x contains ψ, its value is HFL^1 definable. Hence, $\mathcal{S}$ can win the game by choosing this conjunct. If $m < n$, then the approximation stored in f_X is not stable, i.e., there is an HFL^1-definable, and, hence ML-definable set T such that v is in the mth approximation of X applied to the formula defining T, but not in the $m+1$st approximation applied to the formula defining T. Hence, $\mathcal{S}$ can chose the conjunct $\varphi^1_H\, \varphi^1_t$, construct the formula defining T in H and call φ^1_t on this formula, which then evaluates to false.

The Case of HFL^1

Let HFL^1_1 denote the *monadic* fragment of HFL^1, i.e., the set of formulas restricted to types $\bullet$ and $\bullet \to \bullet$. We now define a translation that replaces a greatest-fixpoint definition in an HFL^1_1 formula with one that is equivalent over the class of finite structures and contains a least-fixpoint quantifier and two greatest-fixpoint quantifiers that are not alternating with the least-fixpoint quantifier.

Definition 6.3.2. Let $\varphi_X = \nu(X \colon \bullet \to \bullet).\, \psi_X$ be a greatest-fixpoint definition such that $\psi_X \in \mathrm{HFL}^1_1$. Let $\tau = \bullet \to \bullet$.

Recall the definition of φ'_X. We define φ'_X as

$$\Big(\mu(F_X \colon (\tau) \to \bullet \to \bullet).\, \lambda(f_X \colon \tau).\, \lambda(x \colon \bullet).\\ \big((f_X\, x) \wedge [a]^*(\varphi^1_H\, \varphi^1_t)\big) \vee \big(F_X\, \psi_X[f_X/X]\big)\, x\Big)\, \lambda(z \colon \bullet).\, \mathtt{tt}$$

where

$$\varphi^1_t = \lambda(y \colon \bullet).\, (f_X\, y) \to (\psi_X[f_X/X]\, y)$$

and

$$\begin{aligned}\varphi^1_H = \nu(H \colon (\tau) \to \bullet).\, \lambda(t \colon \bullet \to \bullet).\, & (t\, P) \wedge (t\, \neg P)\\ & \wedge (H\, \lambda(z \colon \bullet)\, t(\langle a \rangle z)) \wedge (H\, \lambda(z \colon \bullet)\, t([a]z))\\ & \wedge (H\, \lambda(z_1 \colon \bullet)\, (H \lambda(z_2 \colon \bullet)\, t\, (z_1 \vee z_2)))\\ & \wedge (H\, \lambda(z_1 \colon \bullet)\, (H \lambda(z_2 \colon \bullet)\, t\, (z_1 \wedge z_2))).\end{aligned}$$

The next two lemmas are dedicated to showing that φ_H^1 does what was explained in the previous section, i.e., apply the test passed to it as an argument to every set defined by an ML formula.

Lemma 6.3.3. *Let ψ be of type $\bullet \to \bullet$. Then, for all LTS $\mathcal{T}$ and all interpretations η, we have that*

$$[\![\varphi_H^1\,\psi]\!]_{\mathcal{T}}^{\eta} \subseteq \bigcap_{\varphi \in \mathsf{ML}} [\![\psi\,\varphi]\!]_{\mathcal{T}}^{\eta}.$$

Proof. Let $\varphi \in \mathsf{ML}$. We have to show that, for all $\psi \in$ HFL, we have that if $v \in [\![\varphi_H^1\,\psi]\!]_{\mathcal{T}}^{\eta}$ then $v \in [\![\psi\,\varphi]\!]_{\mathcal{T}}^{\eta}$. We show the statement by induction over depth(φ). If depth$(\varphi) = 1$ then $\varphi = P$ or $\varphi = \overline{P}$. By unfolding of φ_H^1, we obtain that

$$\begin{aligned}\varphi_H^1\,\psi \equiv &\Big(\lambda t.\,(t\,P) \wedge (t\,\neg P) \wedge (\varphi_H^1\,\lambda z.\,t(\langle a\rangle z)) \wedge (\varphi_H^1\,(\lambda z.\,t([a]z)))\\ &\wedge (\varphi_H^1\,(\lambda z_1.\,(\varphi_H^1\,\lambda z_2.\,t\,(z_1 \vee z_2)))) \wedge (\varphi_H^1\,\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,t\,(z_1 \wedge z_2)))\Big)\,\psi.\end{aligned}$$

Via β-reduction, this is equivalent to

$$\begin{aligned}\varphi_H^1\,\psi \equiv &(\psi\,P) \wedge (\psi\,\neg P) \wedge (\varphi_H^1\,\lambda z.\,\psi(\langle a\rangle z)) \wedge (\varphi_H^1\,(\lambda z.\,\psi([a]z)))\\ &\wedge (\varphi_H^1\,(\lambda z_1.\,(\varphi_H^1\,\lambda z_2.\,\psi\,(z_1 \vee z_2)))) \wedge (\varphi_H^1\,\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,\psi\,(z_1 \wedge z_2))).\end{aligned}$$

By the HFL semantics of conjunctions, we obtain that $v \in [\![\varphi_H^1\,\psi]\!]_{\mathcal{T}}^{\eta}$ implies $v \in [\![\psi\,P]\!]_{\mathcal{T}}^{\eta}$ and $v \in [\![\psi\,\overline{P}]\!]_{\mathcal{T}}^{\eta}$.

Now assume that we have shown the claim for arbitrary ψ and all ML formulas of formula depth n or less. Let φ be such that depth$(\varphi) \leq n+1$. By the same argument as before, we obtain that

$$\begin{aligned}\varphi_H^1\,\psi \equiv &(\psi\,P) \wedge (\psi\,\neg P) \wedge (\varphi_H^1\,\lambda z.\,\psi(\langle a\rangle z)) \wedge (\varphi_H^1\,(\lambda z.\,\psi([a]z)))\\ &\wedge (\varphi_H^1\,(\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,\psi\,(z_1 \vee z_2)))) \wedge (\varphi_H^1\,\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,\psi\,(z_1 \wedge z_2))).\end{aligned}$$

In particular, from $v \in [\![\varphi_H^1\,\psi]\!]_{\mathcal{T}}^{\eta}$, we obtain that

- $v \in [\![\varphi_H^1\,(\lambda z.\,\psi\langle a\rangle z)]\!]_{\mathcal{T}}^{\eta}$,
- $v \in [\![\varphi_H^1\,(\lambda z.\,\psi[a]z)]\!]_{\mathcal{T}}^{\eta}$,
- $v \in [\![\varphi_H^1\,(\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,\psi\,(z_1 \vee z_2)))]\!]_{\mathcal{T}}^{\eta}$, and
- $v \in [\![\varphi_H^1\,(\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,\psi\,(z_1 \wedge z_2)))]\!]_{\mathcal{T}}^{\eta}$.

Since depth$(\varphi) \leq n+1$, we have that $\varphi = \langle a\rangle\varphi'$, respectively $[a]\varphi'$ such that depth$(\varphi') \leq n$, or $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \varphi_1 \wedge \varphi_2$ such that depth$(\varphi_i) \leq n$ for $i \in \{1,2\}$.

In the first case, note that $\lambda z.\,\psi\langle a\rangle z$, respectively $\lambda z.\,\psi[a]z$ match the induction hypothesis, i.e, from $v \in [\![\varphi_t^1\,\lambda z.\,\psi(\langle a\rangle z)]\!]_{\mathcal{T}}^{\eta}$ we obtain that $v \in [\![\lambda z.\,\psi(\langle a\rangle z)\,\varphi'']\!]_{\mathcal{T}}^{\eta}$ for all $\varphi'' \in \mathsf{ML}$ such that depth$(\varphi'') \leq n$. In particular, this holds for φ'. After β-reduction, we obtain $v \in [\![\psi\,\langle a\rangle\varphi]\!]_{\mathcal{T}}^{\eta}$, which is the claim for the case of $\varphi = \langle a\rangle\varphi'$. The case for $\varphi = [a]\varphi'$ is completely symmetric.

Now let $\varphi = \varphi_1 \vee \varphi_2$ such that depth$(\varphi_i) \leq n$ for $i \in \{1,2\}$. From

$$v \in [\![\varphi_H^1\,(\lambda z_1.\,(\varphi_H^1 \lambda z_2.\,\psi\,(z_1 \vee z_2)))]\!]_{\mathcal{T}}^{\eta},$$

we obtain, via the induction hypothesis, that

$$v \in [\![(\lambda z_1.\, (\varphi^1_H \lambda z_2.\, \psi\, (z_1 \vee z_2)))\, \varphi'']\!]^\eta_\mathcal{T}$$

for all $\varphi'' \in \mathsf{ML}$ with $\operatorname{depth}(\varphi'') \leq n$. By choosing $\varphi'' = \varphi_1$, we obtain that

$$v \in [\![(\lambda z_1.\, (\varphi^1_H \lambda z_2.\, \psi\, (z_1 \vee z_2)))\, \varphi_1]\!]^\eta_\mathcal{T}$$

and, via β-reduction, that

$$v \in [\![(\varphi^1_H \lambda z_2.\, \psi\, (\varphi_1 \vee z_2)))\,]\!]^\eta_\mathcal{T}.$$

Another repetition of the argument yields $v \in [\![\varphi^1_H\, \varphi_1 \vee \varphi_2]\!]^\eta_\mathcal{T}$. The case for $\varphi = \varphi_1 \wedge \varphi_2$ is completely symmetric. □

Over finite LTS, the converse of Lemma 6.3.3 also holds.

Lemma 6.3.4. *Let ψ be of type $\bullet \to \bullet$. Then, for all* finite *LTS $\mathcal{T}$ and all interpretations η, we have that*

$$[\![\varphi^1_H\, \psi]\!]^\eta_\mathcal{T} \supseteq \bigcap_{\varphi \in \mathsf{ML}} [\![\psi\, \varphi]\!]^\eta_\mathcal{T}.$$

Proof. By the Kleene Fixpoint Theorem (Thm 2.1.7), φ^1_H is equivalent to some finite approximation H^n with $n \in \mathbb{N}$ defined via

$$\begin{aligned} H^0 &= \lambda t.\, \mathtt{tt} \\ H^{i+1} &= \lambda t.\ (t\, P) \wedge (t\, \overline{P}) \wedge (H^i\, \lambda z.\, t\, \langle a \rangle z) \wedge (H^i\, \lambda z.\, t\, [a] z) \\ &\quad \wedge (H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, t\, (z_1 \vee z_2)))) \wedge (H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, t\, (z_1 \wedge z_2)))). \end{aligned}$$

We now show, by induction over $\mathbb{N} \setminus \{0\}$ that

$$[\![H^i\, \psi]\!]^\eta_\mathcal{T} \supseteq \bigcap_{\varphi \in \mathsf{ML}, \operatorname{depth}(\varphi) \leq i} [\![\psi\, \varphi]\!]^\eta_\mathcal{T}.$$

The claim of the lemma then follows. Note that no ML formulas have formula depth 0. For the base case of $i = 1$, consider

$$\begin{aligned} H^1 &= \lambda t.\ (t\, P) \wedge (t\, \overline{P}) \wedge (H^0\, \lambda z.\, t\, \langle a \rangle z) \wedge (H^0\, \lambda z.\, t\, [a] z) \\ &\quad \wedge (H^0\, (\lambda z_1.\, (H^0\, \lambda z_2.\, t\, (z_1 \vee z_2)))) \wedge (H^0\, (\lambda z_1.\, (H^0\, \lambda z_2.\, t\, (z_1 \wedge z_2)))) \\ &\quad \equiv \lambda t.\ (t\, P) \wedge (t\, \overline{P}) \wedge (\mathtt{tt}) \wedge (\mathtt{tt}) \wedge \mathtt{tt}) \wedge (\mathtt{tt}) \\ &\quad \equiv \lambda t.\, (t\, P) \wedge (t\, \overline{P}). \end{aligned}$$

Clearly $H^1\, \psi \equiv \psi\, P \wedge \psi\, \overline{P}$, which is as claimed.

Now assume that we have shown the claim for $i \geq 1$. Let $\psi \in \mathrm{HFL}$ be arbitrary and consider H^{i+1} which is

$$\begin{aligned} H^{i+1} &= \lambda t.\ (t\, P) \wedge (t\, \overline{P}) \wedge (H^i\, \lambda z.\, t\, \langle a \rangle z) \wedge (H^i\, \lambda z.\, t\, [a] z) \\ &\quad \wedge (H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, t\, (z_1 \vee z_2)))) \wedge (H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, t\, (z_1 \wedge z_2)))). \end{aligned}$$

We have to show that if

$$v \in \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i+1} [\![\psi\, \varphi]\!]^{\eta}_{\mathcal{T}}$$

then $v \in [\![H^{i+1}\, \psi]\!]^{\eta}_{\mathcal{T}}$. By β-reduction and the semantics of conjunctions, this reduces to the problem of showing that

$$\begin{aligned} v \in & [\![\psi\, P]\!]^{\eta}_{\mathcal{T}} \cup [\![\psi\, \overline{P}]\!]^{\eta}_{\mathcal{T}} \cup [\![H^i\, \lambda z.\, \psi\, \langle a \rangle z]\!]^{\eta}_{\mathcal{T}} \cup [\![H^i\, \lambda z.\, \psi\, [a] z]\!]^{\eta}_{\mathcal{T}} \\ & \cup [\![H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, \psi\, (z_1 \vee z_2)))]\!]^{\eta}_{\mathcal{T}} \cup [\![H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, \psi\, (z_1 \wedge z_2)))]\!]^{\eta}_{\mathcal{T}}. \end{aligned}$$

We can show the statement for the conjuncts individually.

- For the first two, membership follows directly from the assumption.
- We have to show that $v \in [\![H^i\, (\lambda z.\, \psi\, \langle a \rangle z)]\!]^{\eta}_{\mathcal{T}}$. By the induction hypothesis, it is enough to show that

  $$v \in \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i} [\![(\lambda z.\, \psi\, \langle a \rangle z)\, \varphi]\!]^{\eta}_{\mathcal{T}}.$$

  Again, using β-reduction, this simplifies to

  $$v \in \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i} [\![\psi\, \langle a \rangle \varphi]\!]^{\eta}_{\mathcal{T}},$$

  and since $\mathrm{depth}(\langle a \rangle \varphi) \leq i + 1$ if $\mathrm{depth}(\varphi) \leq i$, the last membership relation follows from the induction hypothesis.
- The case $v \in [\![H^i\, \lambda z.\, \psi\, \langle a \rangle z]\!]^{\eta}_{\mathcal{T}}$ follows similarly.
- We have to show that $v \in [\![H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, \psi\, (z_1 \vee z_2)))]\!]^{\eta}_{\mathcal{T}}$. Using the same approach as before, it is enough to show that

  $$v \in \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i} [\![(\lambda z_1.\, (H^i\, \lambda z_2.\, \psi\, (z_1 \vee z_2)))\, \varphi]\!]^{\eta}_{\mathcal{T}},$$

  which reduces to

  $$v \in \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i} [\![H^i\, \lambda z_2.\, \psi\, (\varphi \vee z_2)]\!]^{\eta}_{\mathcal{T}},$$

  and, via a second invocation of the argument, to

  $$v \in \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i}\ \bigcap_{\varphi' \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i} [\![\psi\, (\varphi \vee \varphi')]\!]^{\eta}_{\mathcal{T}},$$

  which, again is covered by the induction hypothesis.
- The case $v \in [\![H^i\, (\lambda z_1.\, (H^i\, \lambda z_2.\, \psi\, (z_1 \wedge z_2)))]\!]^{\eta}_{\mathcal{T}}$ is shown similarly.

Since the inclusion

$$[\![H^i\, \psi]\!]^{\eta}_{\mathcal{T}} \supseteq \bigcap_{\varphi \in \mathsf{ML}, \mathrm{depth}(\varphi) \leq i} [\![\psi\, \varphi]\!]^{\eta}_{\mathcal{T}}.$$

holds for all $i \geq 1$, in particular it holds for n. Since the left side is equal for all subsequent natural numbers, the claim of the lemma follows. $\square$

In the next three lemmas, we prove that, in order to know the value of a fixpoint at an HFL_1^1-definable arguments, it is indeed enough to test for stabilization on all ML-definable arguments.

Lemma 6.3.5. *Let φ be an HFL_1^1 formula. Over each finite LTS, φ is equivalent to an HFL_1^1 formula of the form ψ or $\lambda x.\, \psi$ such that ψ does not contain fixpoint binders or lambda abstractions.*

Proof. Let $\mathcal{T}$ be a finite LTS. Since all type lattices over $\mathcal{T}$ are finite, each fixpoint definition in φ is equivalent to a finite unfolding of the fixpoint. By induction over the nesting of fixpoint binders in φ, we obtain a formula φ' that is equivalent to φ over $\mathcal{T}$, i.e., defines the same set or function in the respective type lattice.

Moreover, by repeated application of β-reduction, all lambda abstractions, except a possible one at the front (if φ is not of type $\bullet$), can be removed. Due to strong normalization of the Simply-Typed Lambda Calculus (cf. Thm 2.3.1), this process eventually ends, whence we obtain an equivalent formula of the desired format. $\square$

Note that the equivalence in the lemma above holds just with respect to the given, fixed LTS, i.e., for each finite LTS there might be a different formula of the above form that is equivalent over the LTS in question. Note that equivalent means that the formulas define the same set on the LTS, or the same function. What is *not* meant here is that the formulas are equivalent only on a given vertex of the LTS in question.

Lemma 6.3.6. *Let $\varphi_X = \nu(X : \bullet \to \bullet).\, \psi_X$ be a greatest-fixpoint definition and let ψ be a ground-type formula such that ψ_X and ψ are HFL_1^1. Let $\mathcal{T}, v$ be a finite pointed LTS. Let η be an interpretation that assigns all variables to HFL_1^1-definable sets. Let X^i for $i \in \mathbb{N}$ denote approximations of the semantics of φ_X over $\mathcal{T}$, defined via*

$$X^0 = \lambda(x : \bullet).\, \mathtt{tt}$$
$$X^{i+1} = \psi_X[X^i/X].$$

Then if $[\![X^n\, \psi']\!]^\eta_\mathcal{T} = [\![X^{n+1}\, \psi']\!]^\eta_\mathcal{T}$ for all $\psi' \in \mathrm{HFL}_1^1$, then $[\![X^n\, \psi]\!]^\eta_\mathcal{T} = [\![\varphi_X\, \psi]\!]^\eta_\mathcal{T}$.

Proof. Let n be as described. We show that, in fact $[\![X^{n+1}\, \psi']\!]^\eta_\mathcal{T} = [\![X^{n+2}\, \psi']\!]^\eta_\mathcal{T}$ for all $\psi' \in \mathrm{HFL}_1^1$. By repeating the argument, we obtain that $[\![X^n\, \psi']\!]^\eta_\mathcal{T} = [\![X^m\, \psi']\!]^\eta_\mathcal{T}$ for all $m \geq n$. Since $\mathcal{T}$ is finite, there is m' such that $[\![X^{m'}]\!]^\eta_\mathcal{T} = [\![\varphi_X]\!]^\eta_\mathcal{T}$ due to the Kleene Fixpoint Theorem (Theorem 2.1.7). If X^n agrees with all $X^{n'}$ where $n' \geq n$ on arguments in HFL1, in particular it agrees with $X^{m'}$ on all such arguments. Since ψ is in HFL_1^1, the result follows.

We proceed to show that $[\![X^{n+1}\, \psi']\!]^\eta_\mathcal{T} = [\![X^{n+2}\, \psi']\!]^\eta_\mathcal{T}$ for all $\psi' \in \mathrm{HFL}_1^1$. Note that $X^{n+1} = \psi_X[X^n/X]$ and that $X^{n+2} = \psi_X[X^{n+1}/X]$. We show by induction over the syntax tree of ψ_X that for all subformulas ψ'' of ψ_X, it holds that

- if ψ'' is of ground type, then $[\![\psi''[X^n/X]]\!]^\eta_\mathcal{T} = [\![\psi''[X^{n+1}/X]]\!]^\eta_\mathcal{T}$, and
- if ψ'' is of type $\bullet \to \bullet$ and $T \in [\![\bullet]\!]_\mathcal{T}$ is defined by an HFL_1^1 formula, then $[\![\psi''[X^n/X]]\!]^\eta_\mathcal{T}\, T = [\![\psi''[X^{n+1}/X]]\!]^\eta_\mathcal{T} T$.

The argument now depends on the form of ψ'':

- If ψ'' is a proposition, negated proposition, the claim is immediate.
- If ψ'' is a lambda variable, the claim follows from the assumption on η.
- If ψ'' is a disjunction, a conjunction or a modal formula, the claim follows from a simple invocation of the induction hypothesis.
- If ψ'' is X, then $\psi''[X^n/X] = X^n$, and $\psi''[X^{n+1}/X] = X^{n+1}$. The claim follows from the assumptions on X^n and X^{n+1}.
- If ψ'' is $\psi''_1\,\psi''_2$, then, by the induction hypothesis,

  $$[\![\psi''_2[X^n/X]]\!]^\eta_{\mathcal{T}} = [\![\psi''_2[X^{n+1}/X]]\!]^\eta_{\mathcal{T}}.$$

  Clearly, this set is HFL^1_1-definable. Moreover, by another invocation of the induction hypothesis, we have that $[\![\psi''_1[X^n/X]]\!]^\eta_{\mathcal{T}}$ and $[\![\psi''_1[X^{n+1}/X]]\!]^\eta_{\mathcal{T}}$ agree on all HFL^1_1-definable sets, so the claim follows.

This finishes the proof. □

We now formalize the observation that, over any given finite LTS, every HFL^1_1-definable formula of ground type is equivalent to one in ML.

Lemma 6.3.7. *Let $\mathcal{T}$ be a finite LTS, and let $\varphi \in \mathrm{HFL}^1_1$ be of ground type with no free fixpoint variables. Let η be an interpretation that assigns variables only to HFL^1_1-definable sets. Then there is $\psi \in \mathsf{ML}$ such that $[\![\varphi]\!]^\eta_{\mathcal{T}} = [\![\psi]\!]^\eta_{\mathcal{T}}$.*

Proof. As a first step, there is φ' such that $[\![\varphi']\!]^\eta_{\mathcal{T}} = [\![\varphi]\!]^\eta_{\mathcal{T}}$ and such that φ' does not contain free lambda variables. It is constructed by replacing a lambda variable by the HFL^1_1-formula defining it. Such a formula is necessarily lambda-variable free, so this eliminates free lambda variables. We then apply Lemma 6.3.5 to obtain ψ with $[\![\varphi']\!]^\eta_{\mathcal{T}} = [\![\psi]\!]^\eta_{\mathcal{T}}$. Since ψ has no free lambda variables and contains neither fixpoint binders nor lambda abstraction, it must be in ML. □

Lemma 6.3.8. *Let $\varphi_X = \nu(X\colon \bullet \to \bullet).\,\psi_X$ be a greatest-fixpoint definition with $\psi_X \in \mathrm{HFL}^1_1$. Let φ'_X, φ^1_t and φ^1_H be as defined in Definition 6.3.2. Then, for all formulas $\psi \in \mathrm{HFL}^1_1$, all finite LTS and all interpretations η that assign variables only to HFL^1_1-definable sets, we have that $[\![\varphi_X\,\psi]\!]^\eta_{\mathcal{T}} = [\![\varphi'_X\,\psi]\!]^\eta_{\mathcal{T}}$.*

Proof. Let $\mathcal{T}, v$ be a finite pointed LTS and let η be as in the lemma. Let X^i for $i \in \mathbb{N}$ denote approximations of the semantics of φ_X over $\mathcal{T}$, defined via

$$\begin{aligned} X^0 &= \lambda(x\colon \bullet).\,\mathtt{tt} \\ X^{i+1} &= \psi_X[X^i/X]. \end{aligned}$$

Since $\mathcal{T}$ is finite, we know that φ_X is equivalent to X^n for some $n \in \mathbb{N}$.

We begin the proof by generating a sequence of formulas $\varphi_0, \varphi_1, \ldots, \varphi_n$ in order to show the claim of the lemma. Recall that φ'_X is defined as

$$\Big(\mu(F_X\colon (\bullet \to \bullet) \to \bullet \to \bullet).\,\lambda(f_X\colon \bullet \to \bullet).\,\lambda(x\colon \bullet).\,\big((f_X\,x) \wedge [a]^*(\varphi^1_H\,\varphi^1_t)\big)$$
$$\vee \big(F_X\,\psi_X[f_X/X]\big)\,x\Big)\,\lambda(z\colon \bullet).\,\mathtt{tt},$$

and that

$$\varphi_t^1 = \lambda(y\colon \bullet).\,\big((f_X\, y) \to (\psi_X[f_X/X])\big)\, y,$$

and that φ_H^1 has no free variables.

Since $\lambda z.\mathtt{tt} \equiv X^0$, we can write $\varphi_0 = \varphi'_X\, \psi$ without type annotations as

$$\Big(\mu F_X.\,\lambda f_X.\,\lambda x.\,\big((f_X\, x) \wedge [a]^*(\varphi_H^1\, \varphi_t^1)\big) \vee \big(F_X\, \psi_X[f_X/X]\big)\, x\Big)\, X^0\, \psi.$$

By fixpoint unfolding, we obtain that this is equivalent to

$$\Big(\lambda f_X.\,\lambda x.\,\big((f_X\, x) \wedge [a]^*(\varphi_H^1\, \varphi_t^1)\big) \vee \big(\varphi'_X\, \psi_X[f_X/X]\big)\, x\Big)\, X^0\, \psi$$

and β-reduces to

$$\big((X^0\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^0/f_X])\big) \vee \big(\varphi'_X\, \psi_X[X^0/X]\big)\, \psi,$$

which we denote by φ_1. Note that, in the right disjunct, $\psi_X[X^0/X]$ is X^1. Continuing the unfolding generates the formula φ_i which is

$$\begin{aligned}&\big((X^0\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^0/f_X])\big) \vee \cdots\\ &\qquad\qquad \vee \big((X^i\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^i/f_X])\big) \vee \big(\varphi'_X\, \psi_X[X^i/X]\big)\, \psi.\end{aligned}$$

In particular, since X^{n+1} is equivalent to X^n over $\mathcal{T}$ and under η, we obtain that, after n unfoldings, we generated no additional disjuncts that are not equivalent to one of the disjuncts already generated. Now consider φ_n, which is

$$\begin{aligned}&\big((X^0\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^0/f_X])\big) \vee \cdots\\ &\qquad\qquad \vee \big((X^n\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^n/f_X])\big) \vee \big(\varphi'_X\, \psi_X[X^n/X]\big)\, \psi.\end{aligned}$$

Assume that $v \in [\![\varphi_X\, \psi]\!]_{\mathcal{T}}^{\eta}$. Consider the disjunct $\big((X^n\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^n/f_X])\big)$ in φ_n. Note that, since X^n is the final approximation of φ_X, we have that $v \in [\![X^n\, \psi]\!]_{\mathcal{T}}^{\eta}$. Moreover, using the characterization of φ_H^1 from Lemmas 6.3.3 and 6.3.4, we obtain that over $\mathcal{T}$, we have that $[\![\varphi_H^1\, \varphi_t^1[X^n/f_X]]\!]_{\mathcal{T}}^{\eta}$ is equivalent to

$$\bigcap_{\varphi \in \mathsf{ML}} [\![\varphi_t^1[X^n/f_X]\, \varphi]\!]_{\mathcal{T}}^{\eta},$$

which, by the definition of φ_t^1 resolves to

$$\bigcap_{\varphi \in \mathsf{ML}} [\![(X^n\, \varphi) \to (\psi_X[X^n/X]\, \varphi)]\!]_{\mathcal{T}}^{\eta},$$

and, since X^n is the final approximation, is equivalent to

$$\bigcap_{\varphi \in \mathsf{ML}} [\![(\varphi_X\, \varphi) \to (\psi_X[\varphi_X/X]\, \varphi)]\!]_{\mathcal{T}}^{\eta},$$

which is, by the fixpoint unfolding principle, equivalent to

$$\bigcap_{\varphi \in \mathsf{ML}} [\![(\psi_X[\varphi_X/X]\, \varphi) \to (\psi_X[\varphi_X/X]\, \varphi)]\!]_{\mathcal{T}}^{\eta},$$

and, hence, trivially true everywhere. It follows that

$$v \in \llbracket \big((X^n\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^n/f_X])\big) \rrbracket_{\mathcal{T}}^{\eta}$$

and, hence $v \in \llbracket \varphi'_X \rrbracket_{\mathcal{T}}^{\eta}$.

Conversely, assume that $v \notin \llbracket \varphi_X \rrbracket_{\mathcal{T}}^{\eta}$. Then, we can again generate unfoldings $\varphi_0, \varphi_1, \ldots, \varphi_n$ as above. Consider some disjunct

$$(X^m\, \psi) \wedge [a]^*(\varphi_H^1\, \varphi_t^1[X^m/f_X]).$$

We show that, if $v \in \llbracket [a]^*(\varphi_H^1\, \varphi_t^1[X^m/f_X]) \rrbracket_{\mathcal{T}}^{\eta}$, then $v \notin \llbracket X^m\, \psi \rrbracket_{\mathcal{T}}^{\eta}$ and, hence in either case,

$$v \notin \llbracket (X^m\, \psi) \wedge (\varphi_H^1\, \varphi_t^1[X^m/f_X]) \rrbracket_{\mathcal{T}}^{\eta}.$$

Assume that $v \in \llbracket [a]^*(\varphi_H^1\, \varphi_t^1[X^m/f_X]) \rrbracket_{\mathcal{T}}^{\eta}$. Then, by the characterizations of φ_t^1 in Lemmas 6.3.3 and 6.3.4, we have that every reachable vertex w is such that

$$w \in \bigcap_{\varphi \in \mathsf{ML}} \llbracket (X^m\, \varphi) \rightarrow (\psi_X[X^m/X]\, \varphi) \rrbracket_{\mathcal{T}}^{\eta}.$$

By Lemma 6.3.7, over $\mathcal{T}$, this is actually equivalent to

$$w \in \bigcap_{\varphi \in \mathrm{HFL}_1^1} \llbracket (X^m\, \varphi) \rightarrow (\psi_X[X^m/X]\, \varphi) \rrbracket_{\mathcal{T}}^{\eta},$$

and, by Lemma 6.3.6, we obtain that $\llbracket X^m\, \varphi \rrbracket_{\mathcal{T}}^{\eta} = \llbracket \varphi_X\, \varphi \rrbracket_{\mathcal{T}}^{\eta}$ for all $\varphi \in \mathrm{HFL}_1^1$. In particular, this holds for ψ, and, since $v \notin \llbracket \varphi_X \rrbracket_{\mathcal{T}}^{\eta}$, we have that $v \notin \llbracket X^n\, \psi \rrbracket_{\mathcal{T}}^{\eta}$ as desired.

Since m was arbitrary, we have that for no disjunct in an arbitrarily large unfolding of φ'_X, the semantics of the disjunct over $\mathcal{T}$ contains v. However, since $\mathcal{T}$ is finite[5], φ'_X is equivalent to some unfolding with the rightmost disjunct containing the fixpoint itself deleted. Hence, $v \notin \llbracket \varphi'_X \rrbracket_{\mathcal{T}}^{\eta}$. $\square$

Of course, the translation defined in Definition 6.3.2 only allows us to rewrite *one* greatest-fixpoint definition into a least-fixpoint definition with two hidden greatest fixpoints that, however, occur in an alternation-free manner, i.e., those in φ_H^1 and in $[a]^*(\ldots)$. Extending this translation such that an arbitrary HFL_1^1-formula can be rewritten into an alternation-free HFL^2-formula is not straightforward. For example, the construction can not be chained in the sense that one can rewrite all greatest fixpoints, starting, e.g., from an innermost one. There are two reasons for this: The first one is that, obviously, the first such invocation of the translation introduces subformulas that are in HFL^2 and not in HFL_1^1. This reason can be argued away by relaxing the above definitions to work not only for HFL_1^1-formulas, but also for those that are equivalent to such a formula, where appropriate. The second reason is more important: Using the translation more than once can introduce unwanted fixpoint alternation between the greatest fixpoints occurring in φ_H^1 and $[a]^* \ldots$ on the one hand, and fixpoint variables occurring freely in the defining formula of another fixpoint, which occurs below $[a]^*(\ldots)$.

[5]Finiteness is not actually required here, since the argument carries over to transfinite ordinals just as easily.

Another approach to define a translation that yields an alternation-free HFL^2-formula for each HFL$^1_1$-formula is to not chain the approximations of different fixpoints sequentially, but rather in a parallel fashion, i.e., the approximants to all fixpoints in a formula are computed in parallel. However, this comes with its own challenges, as the defining formula of a fixpoint can contain occurrences of other fixpoints, both freely and as the defining formula of the fixpoint in question. Managing these occurrences in the correct way is not straightforward.

What is not problematic is an extension of the translation from HFL$^1_1$ to full HFL^1. Extending φ^1_t and φ^1_H to multiple arguments is not difficult, but just makes these formulas even more unwieldy.

The Case of HFL$^2_1$

The approach of the previous section to rewrite a single fixpoint into one of the opposing polarity generalizes to HFL$^2_1$, the fragment of HFL^2 restricted to monadic types, i.e., those with at most one argument. The idea follows the same pattern as the proof of Definition 6.3.2. Due to the length of the arguments involved, we do not duplicate it here. The cornerstone of the argument is the observation that, similarly to how each ground-type HFL$^1_1$-formula is equivalent to one in ML over any given finite LTS, any HFL$^2_1$ formula of type $\bullet \to \bullet$ is equivalent to one of the form $\lambda(x\colon \bullet).\, \varphi$ where φ can be derived from the grammar

$$\varphi ::= P \mid \overline{P} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid x,$$

where the term x does not range over a set of variables, but is restricted to just the one variable bound in $\lambda x.\, \varphi$. The argument is the same as in Lemma 6.3.7: Use fixpoint unfolding and β-reduction.

The translation of a formula $\varphi_X = \nu(X\colon (\bullet \to \bullet) \to \bullet).\, \lambda(x\colon \bullet \to \bullet).\, \psi_X$ then is defined as

$$\Big(\mu(F_X\colon \tau \to \bullet).\, \lambda(f_X\colon \tau).\, \lambda(x\colon \tau').\, \big((f_X\, x) \wedge (\varphi^2_t\, \varphi^2_H)\big) \\ \vee \big(F_X\, \psi_X[f_X/X]\big)\, x\Big)\, \lambda(y\colon \tau').\, \mathtt{tt}$$

where $\tau' = \bullet \to \bullet$ and $\tau = (\tau') \to \bullet$ and and

$$\varphi^2_t = \lambda(y'\colon \bullet \to \bullet).\, (f_X\, x) \to [a]^*(\psi_X[f_X/X]\, x),$$

and $\varphi^2_H = \nu\nu(H\colon (\tau \to \bullet).\psi^2_H$ where ψ^2_H is defined as

$$\begin{aligned}\lambda(t\colon \tau).\, &(t\,(\lambda(z\colon \bullet).\, P)) \wedge (t\,(\lambda(z\colon \bullet).\, \overline{P})) \wedge (t\,((z\colon \bullet).\, z)) \\ &\wedge \big(H\, \lambda(f\colon \bullet \to \bullet).\, t(\lambda(z\colon \bullet).\, \langle a \rangle (f\, z))\big) \\ &\wedge \big(H\, \lambda(f\colon \bullet \to \bullet).\, t(\lambda(z\colon \bullet).\, [a](f\, z))\big) \\ &\wedge \big(H_2\, \lambda(f_1\colon \bullet \to \bullet).\, (H\, \lambda(f_2\colon \bullet \to \bullet).\, t\,(\lambda(z\colon \bullet).\, (f_1\, z) \vee f_2\, z))\big) \\ &\wedge \big(H\, \lambda(f_1\colon \bullet \to \bullet).\, (H\, \lambda(f_2\colon \bullet \to \bullet).\, t\,(\lambda(z\colon \bullet).\, (f_1\, z) \wedge f_2\, z))\big).\end{aligned}$$

6.3.2 Further Extensions

The polarity switching approach discussed in the previous section for monadic HFL^1 and HFL^2 does not generalize to all monadic HFL^k without further adaption. The reason for that is failure of the argument that each formula on the candidate set to test on for stabilization is equivalent to a very simple one. For the cases of order 1 and 2, we invoked β-reduction to make sure that we only had to test on formulas that contain no lambda abstraction. However, consider the case of, e.g., an order-4 fixpoint $\nu(X\colon (((\bullet \to \bullet) \to \bullet) \to \bullet) \to \bullet).\, \lambda(x\colon ((\bullet \to \bullet) \to \bullet) \to \bullet).\, \varphi_X$.

In order to test for stabilization of a fixpoint of this kind, we have to verify that it is stable on all suitable objects in $[\![((\bullet \to \bullet) \to \bullet) \to \bullet]\!]$. But not every formula in this type space is equivalent to one not containing lambda abstraction. In fact, a closed formula of the form $\lambda(f\colon (\bullet \to \bullet) \to \bullet).\, \psi$ is either trivial since f does not occur in it, or it must contain a lambda abstraction as an operand to f, which is of type $(\bullet \to \bullet) \to \bullet$. It contains a lambda abstraction since HFL does not allow to construct objects of type $\bullet \to \bullet$ without either free variables or lambda abstraction. However, if we allow lambda abstraction in the set of formulas to test on, e.g., in order to cater to the problem just described, we potentially have to deal with formulas of the form $\lambda(f\colon (\bullet \to \bullet) \to \bullet).\, (f\, \lambda(x\colon \bullet).\, (f\, \lambda(y\colon \bullet).\, f\, \lambda(\bullet\colon z).\, \dots))$, i.e., we have to deal with formulas with a potentially unbounded number of variables. A stronger analysis of the set of formulas required to test on for stabilization is necessary to reduce this problem back to the case of a set of formulas that can be enumerated in HFL itself. Of course, it might also be possible that an extension beyond orders 2 or 3 is not possible at all, due to effects that occur only at certain type levels. For example, equivalence of Idealized Algol becomes undecidable at order 4 [71], while unification becomes undecidable at order 2 [38].

Chapter 7

Conclusion

7.1 Summary

We have advanced the understanding of the interplay between extremal fixpoints and higher-order constructions. APKA represent the first model of operational semantics for HFL that captures the denotational counterpart over the class of all structures and without restrictions. In contrast to the model-checking game exhibited in Chapter 3, the state space of an APKA remains finite to an extent. It is only the environment component that grows in a potentially unbounded manner, whence arguments on e.g., the type of a given configuration during a run of an APKA can still be made. Such a finite state space is useful e.g., in the setting of Section 6.2.1 and Section 6.2.2, where it allows us to encode the run of an APKA such the winner of the acceptance game can be decided by another APKA.

The extension of the theory of unfolding trees beyond FLC has furthered the understanding of the intricacies of recursion in the context of HFL and what does and does not constitute actual infinite behavior. In particular, it is not a priori obvious that exactly one infinite path must exist in an unfolding tree of a play of an APKA. The observation that such a unique path *does* exist can be understood as a proof that the infinitary behavior of HFL is almost exclusively due to fixpoint recursion, which nicely fits with the strong normalization property of the Simply-Typed Lambda Calculus. In other words, even though HFL can be challenging, the interaction of fixpoint recursion and a simply-typed lambda calculus does not allow to completely turn the fundamentals of the latter on their head – if higher-order effects have been normalized away (in an unfolding tree), what remains is pure fixpoint recursion.

The analysis of settings where the interplay between fixpoint recursion and higher-order effects is limited has produced some surprising results. While it is not necessarily surprising that the model-checking problem of HFL becomes easier when restricted to tail-recursive formulas, the extent to which the richness of the interaction between fixpoint recursion and the lambda-calculus parts of HFL collapses in the setting of simple APKA is notable. This is contrasted by the observation that the other side of the coin, tail-recursive formulas, still enjoy high expressive power, suggesting that expressive power in simple APKA, respectively tail-recursive formulas, is restricted in a specific way. This might be seen as a complementary observation to the observation in the previous paragraph: After controlling for higher-order effects, the remaining expressive power is due to fixpoint recursion.

7.2 Further Research

The acceptance condition of APKA justifies further research. As we have seen in Section 6.2.1, whether a configuration is on the infinite path of the unfolding tree of a play can, in general, not be decided until after the play has concluded. Since a configuration is on the infinite path exactly if it exhibits infinite recursion, this means that, generally, in operational semantics of HFL it can remain undecided whether an occurrence of a fixpoint exhibits infinite recursion or not for long parts of the play. Hence, a different acceptance condition is unlikely to grasp this in a fundamentally better way. However, perhaps it is possible to find further fragments of HFL beyond the order-1-fragment and simple APKA where there is at least a one-sided condition on whether some occurrence of a fixpoint configuration is relevant to the acceptance condition or not. For example, it is likely that the order-1-fragment does not behave differently if we add the ability to have functions of the form $\lambda(x\colon \bullet).\, \langle a\rangle^i x$ as arguments, even though this technically is order-2 behavior.

Another area of research comprises the relationship between tail-recursive formulas and simple APKA, given the apparent mismatch between the two notions. As already proposed in Conjecture 6.2.31, it is likely that the fixpoint alternation hierarchy is strict for tail-recursive formulas with fixed recursion depth, and that an adaption of Arnold's proof is possible for this setting. Moreover, the rather strict setting of APKA might introduce artificial difficulties here in the following sense: Since lambda abstraction is implicit in APKA and, hence always occurs together with a fixpoint state, this somewhat limits the possibilities for simple APKA, since any lambda abstraction, even if it is of the form $\lambda(x\colon \bullet).\, \langle a\rangle^i x$ or comparable, requires an accompanying fixpoint state that is subject to the partition making the APKA simple. On the other hand, lambda abstraction can be used much more freely in tail-recursive formulas, since lambda abstraction does not produce free fixpoint variables. It might be possible to designate some fixpoint states of a simple APKA as non-recursive and allow them to be used more freely. Another approach would be to depart from the relatively standardized form of APKA and re-consider the variant in [16], which was designed along the syntax tree of an HFL formula and, hence, may contain unrestricted use of lambda abstraction.

Loosening the standardized form of APKA might be an interesting idea in general. The coupling of lambda abstraction and fixpoint unfolding was introduced to make the arguments around the acceptance condition, as well as the translations to and from HFL, more accessible. Now that a solid theory of the operational behavior of HFL is in place, these restrictions might be no longer necessary.

The penultimate research item presented here concerns HORS model-checking. In HORS model-checking, the higher-order effects of HORS are in some sense decoupled from the extremal fixpoint behavior of PA since both appear on opposite sides of the model-checking problem. We have seen in the context of simple APKA that an artificial decoupling for the HFL setting has dramatic effects. Given that HFL model-checking and HORS model-checking against PA are inter-reducible, it might be interesting to look at the image of simple APKA under the translation, considering that this might unearth certain combinations of HORS and PA that are themselves conceptually simpler.

Finally the results in Section 6.3 open up a completely new area of research. The first question here concerns whether the results do actually lead to a complete collapse result. Preliminary research suggests that this is the case, but a more thorough verification is needed. In particular, the understanding of alternation-freeness in the higher-order setting is underdeveloped. However, we can give the following conjecture:

Conjecture 7.2.1. *The $\mathcal{L}_\mu$-alternation hierarchy collapses into the alternation-free fragment of* HFL^1*. Also, the* HFL^1 *alternation hierarchy collapses into the alternation-free fragment of* HFL^2*. Moreover, the* HFL^2 *alternation hierarchy collapses into the alternation-free fragment of* HFL^3*.*

Even if a full collapse of the alternation hierarchy is not obtainable, the polarity switching technique is potentially useful and the extent to which it can be developed should be mapped out. Moreover, while the discussion in Section 6.3 shows that the naive formula enumeration trick is not enough at sufficiently high order, closer inspection of the stabilization criteria appears to yield that it is enough to test for stabilization on a small enough subset of formulas that can be enumerated. Additionally, a proper characterization of the alternation-free fragment of HFL is also a research target.

The last question that comes up in this context is that of fixpoint conversion. The arguments made in Section 6.3 invoke finiteness of the LTS in question, but, in fact, just require that fixpoints of a given order stabilize at a finite approximation. This is a property that is not restricted to finite LTS. In fact, we conjecture that there is a proper hierarchy of structure classes such that

$$\mathbb{T}^0_{\mathsf{fin}} \supseteq \mathbb{T}^1_{\mathsf{fin}} \supseteq \cdots \supseteq \bigcap_{i \in \mathbb{N}} \mathbb{T}^i_{\mathsf{fin}} \supseteq \mathbb{T}^{\sim}_{\mathsf{fin}} .$$

where $\mathbb{T}^i_{\mathsf{fin}}$ is the class of all structures on which fixpoint definitions of order i or less stabilize after finitely many, but not necessarily uniformly finitely many approximations, and where $\mathbb{T}^{\sim}_{\mathsf{fin}}$ is the class of structures with finite bisimulation quotient, over which it is easy to see that all fixpoint definitions stabilize after finitely many approximations.

Bibliography

[1] Luca Alberucci and Alessandro Facchini. The modal μ-calculus hierarchy over restricted classes of transition systems. *The journal of symbolic logic*, 74(4):1367–1400, 2009.

[2] Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2988 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2004.

[3] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004.

[4] André Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.

[5] Roland Axelsson and Martin Lange. Model checking the first-order fragment of higher-order fixpoint logic. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2007.

[6] Roland Axelsson, Martin Lange, and Rafal Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3(2), 2007.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[8] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922.

[9] Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types.* Perspectives in logic. Cambridge University Press, 2013.

[10] Arnold Beckmann. Exact bounds for lengths of reductions in typed λ-calculus. *The Journal of Symbolic Logic*, 66(3):1277–1285, 2001.

[11] Dietmar Berwanger and Erich Grädel. Fixed-point logics and solitaire games. *Theory Comput. Syst.*, 37(6):675–694, 2004.

[12] Patrick Blackburn, Maarten De Rijke, and Yde Venema. Modal logic (cambridge tracts in theoretical computer science), 2002.

[13] Julian C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theor. Comput. Sci.*, 195(2):133–153, 1998.

[14] Julian C. Bradfield. Fixpoint alternation: Arithmetic, transition systems, and the binary tree. *ITA*, 33(4/5):341–356, 1999.

[15] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. C-shore: a collapsible approach to higher-order verification. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 13–24. ACM, 2013.

[16] Florian Bruse. Alternating parity krivine automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014.

[17] Florian Bruse. Alternation is strict for higher-order modal fixpoint logic. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016.*, volume 226 of *EPTCS*, pages 105–119, 2016.

[18] Florian Bruse, Oliver Friedmann, and Martin Lange. On guarded transformation in the modal μ-calculus. *Logic Journal of the IGPL*, 23(2):194–216, 2015.

[19] Florian Bruse, Martin Lange, and Étienne Lozes. Collapses of fixpoint alternation hierarchies in low type-levels of higher-order fixpoint logic. unpublished.

[20] Florian Bruse, Martin Lange, and Étienne Lozes. The complexity of model-checking the tail-recursive fragment of higher-order modal fixpoint logic. submitted.

[21] Florian Bruse, Martin Lange, and Étienne Lozes. Space-efficient fragments of higher-order fixpoint logic. In Matthew Hague and Igor Potapov, editors, *Reachability Problems - 11th International Workshop, RP 2017, London, UK, September 7-9, 2017, Proceedings*, volume 10506 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017.

[22] J Richard Büchi. On a decision method in restricted second order arithmetic. In *The Collected Works of J. Richard Büchi*, pages 425–435. Springer, 1990.

[23] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th Annual ACM SIGACT Symp. on Theory of Computing, STOC'17*, pages 252–263. ACM, 2017.

[24] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[25] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[26] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

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

[28] Giovanna D'Agostino and Giacomo Lenzi. On the μ-calculus over transitive and finite transitive frames. *Theor. Comput. Sci.*, 411(50):4273–4290, 2010.

[29] Anuj Dawar and Martin Otto. Modal characterisation theorems over special classes of frames. *Annals of Pure and Applied Logic*, 161(1):1–42, 2009.

[30] Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

[31] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 996–1072. Elsevier and MIT Press, New York, USA, 1990.

[32] E. A. Emerson. *Automated Temporal Reasoning about Reactive Systems*, volume 1043 of *LNCS*, pages 41–101. Springer, New York, NY, USA, 1996.

[33] E. Allen Emerson. Uniform inevitability is tree automaton ineffable. *Inf. Process. Lett.*, 24(2):77–79, 1987.

[34] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991.

[35] E Allen Emerson, Charanjit S Jutla, and A Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258(1-2):491–522, 2001.

[36] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 267–278. IEEE Computer Society, 1986.

[37] O. Friedmann and M. Lange. Solving parity games in practice. In *Proc. 7th Int. Symp. on Automated Technology for Verification and Analysis, ATVA'09*, volume 5799 of *LNCS*, pages 182–196, 2009.

[38] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

[39] George Grätzer. *General lattice theory.* Springer Science & Business Media, 2002.

[40] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 60–65. ACM, 1982.

[41] Julian Gutierrez, Felix Klaedtke, and Martin Lange. The μ-calculus alternation hierarchy collapses over structures with restricted connectivity. *Theor. Comput. Sci.*, 560:292–306, 2014.

[42] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008.

[43] David Harel, Amir Pnueli, and Jonathan Stavi. Propositional dynamic logic of nonregular programs. *J. Comput. Syst. Sci.*, 26(2):222–243, 1983.

[44] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[45] Gérard Huet. *Résolution d'équations dans les langages d'ordre 1, 2, ..., ω.* PhD thesis, Université Paris VII, 09 1976.

[46] Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

[47] David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[48] Thomas Jech. *Set theory, Second Edition.* Perspectives in Mathematical Logic. Springer, 1997.

[49] Neil D. Jones. The expressive power of higher-order types or, life without CONS. *J. Funct. Program.*, 11(1):5–94, 2001.

[50] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Proc. 17th Ann. Symp. on Theoretical Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[51] M. Jurdzinski and R. Lazic. Succinct progress measures for solving parity games. In *Proc. 32nd ACM/IEEE Symp. on Logic in Computer Science, LICS'17*, pages 1–9. IEEE, 2017.

[52] Roope Kaivola. Axiomatising linear time mu-calculus. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings*, volume 962 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 1995.

[53] Stephen Cole Kleene. On notation for ordinal numbers. *Journal of Symbolic Logic*, 3(4):150–155, 1938.

[54] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order push-down trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

[55] Naoki Kobayashi, Étienne Lozes, and Florian Bruse. On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 246–259. ACM, 2017.

[56] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009.

[57] Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe. Higher-order program verification via HFL model checking. In Amal Ahmed, editor, *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10801 of *Lecture Notes in Computer Science*, pages 711–738. Springer, 2018.

[58] Dénes König. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math.(Szeged)*, 3(2-3):121–130, 1927.

[59] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[60] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[61] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

[62] Martin Lange. The alternation hierarchy in fixpoint logic with chop is strict too. *Inf. Comput.*, 204(9):1346–1367, 2006.

[63] Martin Lange and Étienne Lozes. Capturing bisimulation-invariant complexity classes with higher-order modal fixpoint logic. In Josep Díaz, Ivan Lanese, and Davide Sangiorgi, editors, *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*, pages 90–103. Springer, 2014.

[64] Martin Lange, Étienne Lozes, and Manuel Vargas Guzmán. Model-checking process equivalences. *Theor. Comput. Sci.*, 560:326–347, 2014.

[65] Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 639–648. ACM, 2018.

[66] Christof Löding, P. Madhusudan, and Olivier Serre. Visibly pushdown games. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 408–420. Springer, 2004.

[67] Étienne Lozes. A type-directed negation elimination. In Ralph Matthes and Matteo Mio, editors, *Proceedings Tenth International Workshop on Fixed Points in Computer Science, FICS 2015, Berlin, Germany, September 11-12, 2015.*, volume 191 of *EPTCS*, pages 132–142, 2015.

[68] Robert S. Lubarsky. μ-definable sets of integers. *J. Symb. Log.*, 58(1):291–313, 1993.

[69] Radu Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2280 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2002.

[70] Markus Müller-Olm. A modal fixpoint logic with chop. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 510–520. Springer, 1999.

[71] Andrzej S. Murawski. On program equivalence in languages with ground-type references. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*, page 108. IEEE Computer Society, 2003.

[72] Damian Niwinski. Fixed point characterization of infinite behavior of finite-state systems. *Theor. Comput. Sci.*, 189(1-2):1–69, 1997.

[73] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006.

[74] Luke Ong. Higher-order model checking: An overview. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 1–15. IEEE Computer Society, 2015.

[75] Christos H Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[76] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[77] Michael O Rabin. Weakly definable relations and special automata. In *Studies in Logic and the Foundations of Mathematics*, volume 59, pages 1–23. Elsevier, 1970.

[78] Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014.

[79] Marco Sälzer. *Neededness Analysis for Model Checking Properties Defined by Order-*2 *Fixpoints*. Bachelor's thesis, Universität Kassel, 2018.

[80] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[81] Sven Schewe. Solving parity games in big steps. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2007.

[82] Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, USA, October 6-8, 1965*, pages 179–190. IEEE Computer Society, 1965.

[83] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theor. Comput. Sci.*, 89(1):161–177, 1991.

[84] Robert S Streett and E Allen Emerson. The propositional mu-calculus is elementary. In *International Colloquium on Automata, Languages, and Programming*, pages 465–472. Springer, 1984.

[85] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

[86] Taku Terao and Naoki Kobayashi. A zdd-based efficient higher-order model checking algorithm. In Jacques Garrigue, editor, *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*, pages 354–371. Springer, 2014.

[87] Tom van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, volume 10805 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2018.

[88] P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker, Inc., 1997.

[89] Moshe Y Vardi and Pierre Wolper. Yet another process logic. In *Workshop on Logic of Programs*, pages 501–512. Springer, 1983.

[90] Mahesh Viswanathan and Ramesh Viswanathan. A higher order modal fixed point logic. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2004.

[91] Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical computer science*, 275(1-2):311–346, 2002.

[92] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Index