Alternation Is Strict For Higher-Order Modal Fixpoint Logic

Florian Bruse

Universität Kassel
Kassel, Germany
`florian.bruse@uni-kassel.de`

We study the expressive power of Alternating Parity Krivine Automata (APKA), which provide operational semantics to Higher-Order Modal Fixpoint Logic (HFL). APKA consist of ordinary parity automata extended by a variation of the Krivine Abstract Machine. We show that the number and parity of priorities available to an APKA form a proper hierarchy of expressive power as in the modal μ-calculus. This also induces a strict alternation hierarchy on HFL. The proof follows Arnold's (1999) encoding of runs into trees and subsequent use of the Banach Fixpoint Theorem.

1 Introduction

Parity automata provide popular operational semantics for the modal μ-calculus and, hence, for all regular properties over trees. They are equivalent to most other acceptance modes with the exception of Büchi automata [13]. However, since parity automata can only express regular properties, extending their expressive power, or extending them to cover stronger logics, is the subject of ongoing research. For example, visibly pushdown automata [1] allow the addition of a limited pushdown stack but tie the stack operations to different and disjoint parts of the alphabet.

In this paper, we revisit our previous work on extending parity automata by a variant of the Krivine Abstract Machine [9], which incorporates a simply typed lambda calculus into the semantics of the automaton model. The resulting *Alternating Parity Krivine Automata* (APKA) yield operational semantics for Higher-Order Modal Fixpoint Logic (HFL) [14]. The acceptance condition of APKA is a stair parity condition over an acceptance game. The stair parity condition resembles that of visibly pushdown automata, but it is not tied to any alphabet symbols or tree labels, but rather emerges via the bookkeeping done by the Krivine Machine part. This automaton model is very expressive: Properties such as uniform inevitability or the presence of a given property in a level that is a power of two are easily expressible. This expressive power comes at a price, since emptiness of APKA, which is equivalent to satisfiability of HFL-formulae, is undecidable.

A key improvement over the variant of APKA presented in [5] is that in this paper, the state space of the automaton is not restricted to a tree-like structure inherited from HFL-formulae, but can take the form of any graph, just like an ordinary parity automaton is less restricted in structure than a formula of the modal μ-calculus. Since in the new variant of APKA, precedence between states representing different fixpoints can not be inferred from their position in a syntax tree, it is given explicitly via a parity labeling of states. This has the advantage that the alternation class of an automaton, or that of any equivalent formula, can be defined via the number of its priorities, while for formulae, alternation can be hard to gauge syntactically. Already for the modal μ-calculus, syntactic criteria to define alternation classes can be quite complex [6, 12]. On the automaton side of things, however, characterization via the number of priorities makes things much easier. Translations from APKA into HFL and vice versa are readily available and any alternation hierarchy for APKA induces an alternation hierarchy on HFL. This settles the question posed in [5] on how to properly define alternation classes for HFL.

We find that for APKA, adding more priorities increases expressive power. The original strictness result for parity automata has a beautiful proof [2] involving the Banach Fixpoint Theorem, which also has been adapted to Fixpoint Logic with Chop FLC [10]. Our strictness proof proceeds in a similar manner: Given an infinite binary tree and an APKA of suitable vocabulary, we construct another infinite binary tree which encodes the acceptance game of the run of the APKA. Given a vocabulary tailored to a specific alternation class, we construct an automaton which accepts such a game tree if and only if the original automaton accepts the original tree. This operation induces a contraction in the complete metric space of infinite binary trees, which, by the Banach Fixpoint Theorem, has a fixpoint. We show that on this fixpoint, no automaton with less priorities or with the same amount of priorities, but flipped parity, can be equivalent to the given meta-automaton.

While strictness of the alternation hierarchy for APKA and, hence, for HFL is not unexpected, such a result is not obvious. It is well known that adding more priorities to a parity automaton or more Rabin pairs to a Rabin automaton increases their expressive power, just as extra fixpoint alternation in the modal μ-calculus does [4]. However, adding extra fixpoint nesting does not always yield more expressive power: The Immerman-Vardi Theorem entails that, over finite ordered structures, first-order logic with least and greatest fixpoints is as strong as first-order logic with only one least fixpoint. Also the alternation hierarchy of the modal μ-calculus itself collapses to the alternation-free fragment over certain classes of structures, for example the class of infinite words [8] and, more generally, classes of structures with restricted connectivity [7]. Preliminary work also shows that alternation for HFL collapses over *finite* structures. It should also be noted that, just like with Fixpoint Logic with Chop [10], formulas that are hard for alternation classes for the modal μ-calculus are not necessary suitable candidates for higher-order logics. This is because these formulas are not designed for the higher-order features of HFL.

The plan of the paper is as follows: In Section 2, we define APKA and their acceptance condition for infinite binary trees. We have a look at their relation to HFL in Section 3. In the following section, we define alternation classes and present a class of trees that encode runs of APKA from a given alternation class. For each alternation class we also construct meta-automata that accept such a tree encoding a run if and only if the run was accepting. This allows us to prove strictness of the alternation hierarchy. The paper closes with a brief discussion of important points.

2   Alternating Parity Krivine Automata

Note that we previously defined APKA differently. This work supersedes earlier definitions in [5]. For ease of exposition, and since the alternation hierarchy argument is developed over the class of fully infinite binary trees, we only consider automata over labeled fully infinite binary trees. The concept of APKA extends naturally to trees of unrestricted branching factor and, or any class of Kripke structures

Fix some set $\mathscr{P}$ of propositions. An infinite binary tree with labels in $\mathscr{P}$ (just tree or $\mathscr{P}$-tree from now on) is given by a function $\mathscr{T}$ from the set $\{0,1\}^*$ of all $\{0,1\}$ words into $2^{\mathscr{P}}$. The root of the tree is identified with ε and the left and right successors of $t \in \{0,1\}^*$ are $t0$ and $t1$, respectively. We say that $P \in \mathscr{P}$ holds at t (written $\mathscr{T},t \models P$) if $P \in \mathscr{T}(t)$. The pair $\mathscr{T},t$ refers to the subtree induced by t.

Simple types are defined inductively via $\tau ::= \mathrm{Pr} \mid \tau \to \tau$. We often refer to Pr as *ground type*. The operator $\to$ is right-associative, so any type can be written as $\tau_1 \to \cdots \to \tau_n \to \mathrm{Pr}$. The order ord is defined inductively via $\mathrm{ord}(\mathrm{Pr}) = 0$ and $\mathrm{ord}(\tau_1 \to \cdots \to \tau_n \to \mathrm{Pr}) = \max(\mathrm{ord}(\tau_1),\ldots,\mathrm{ord}(\tau_n)) + 1$. The set of types is partially ordered via $\tau, \tau' < \tau \to \tau'$. The intended semantics for the ground type over a tree $\mathscr{T}$ is a set of subtrees of $\mathscr{T}$, the intended semantics for a type of the form $\tau_1 \to \cdots \to \tau_n \to \mathrm{Pr}$ is that of a monotone function consuming arguments of types $\tau_1,\ldots,\tau_n$ and returning a set of subtrees of $\mathscr{T}$.

Figure 1: Typing Rules for APKA-transition relations.

$$\frac{}{\Sigma \vdash P \colon \mathrm{Pr}} \qquad \frac{}{\Sigma \vdash \neg P \colon \mathrm{Pr}} \qquad \frac{}{\Sigma \vdash f_j^i \colon \tau_j^i} \qquad \frac{}{\Sigma \vdash X_i \colon \tau_1^i \to \cdots \to \tau_{n_{X_i}}^i \to \mathrm{Pr}} \qquad \frac{\Sigma \vdash \varphi \colon \mathrm{Pr}}{\Sigma \vdash \Diamond_R \varphi \colon \mathrm{Pr}}$$

$$\frac{\Sigma \vdash \varphi \colon \mathrm{Pr}}{\Sigma \vdash \Box_R \varphi \colon \mathrm{Pr}} \qquad \frac{\Sigma \vdash \varphi_1 \colon \mathrm{Pr} \qquad \Sigma \vdash \varphi_2 \colon \mathrm{Pr}}{\Sigma \vdash \varphi_1 \vee \varphi_2 \colon \mathrm{Pr}} \qquad \frac{\Sigma \vdash \varphi_1 \colon \mathrm{Pr} \qquad \Sigma \vdash \varphi_2 \colon \mathrm{Pr}}{\Sigma \vdash \varphi_1 \wedge \varphi_2 \colon \mathrm{Pr}}$$

$$\frac{\Sigma \vdash \varphi \colon \tau \to \tau' \qquad \Sigma \vdash \psi \colon \tau}{\Sigma \vdash (\varphi \, \psi) \colon \tau'}$$

2.1 Definition

Fix a finite set $\mathscr{X} = \{X_1, \ldots, X_n\}$ of states, or fixpoint variables, and a finite set $\mathscr{F}$ which is the disjoint union $\bigcup_n \mathscr{F}_{X_n}$ of lambda variables, where $\mathscr{F}_{X_i} = \{f_1^{X_i}, \ldots, f_{n_i}^{X_i}\}$.

For each $1 \le i \le n$, let φ_i be derived from the grammar

$$\varphi_i ::= P \mid \neg P \mid \Diamond \varphi_i \mid \Box \varphi_i \mid \varphi_i \vee \varphi_i \mid \varphi_i \wedge \varphi_i \mid f \mid X \mid (\varphi_i \, \varphi_i)$$

where $P \in \mathscr{P}$, $X \in \mathscr{X}$ and $f \in \mathscr{F}_{X_i}$.

An *Alternating Parity Krivine Automaton* (APKA) of index m and order k is a five-tuple the form $(\mathscr{X}, \Lambda, X_{\mathrm{init}}, \delta, (\tau_X)_{X \in \mathscr{X}})$ where $\mathscr{X}$ is as above, $X_{\mathrm{init}} \in \mathscr{X}$ is the initial state, $\Lambda \colon \mathscr{X} \to \{1, \ldots, m\}$ or $\Lambda \colon \mathscr{X} \to \{0, \ldots, m-1\}$ labels each fixpoint variable with a priority, the $\tau_X = \tau_1^X \to \cdots \to \tau_{n_X}^X \to \mathrm{Pr}$ of order at most k specify the types of the fixpoints, the type of the initial state $\tau_{X_{\mathrm{init}}}$ is Pr, and δ is the transition relation that maps X_i to φ_i and is such that $\emptyset \vdash \delta(X) \colon \mathrm{Pr}$ for each $X \in \mathscr{X}$, according to the typing rules reproduced in Figure 1. The state space of the automaton is $\mathscr{Q} = \mathscr{X} \cup \bigcup_{X \in \mathscr{X}} \mathrm{sub}(\delta(X))$, where $\mathrm{sub}(\psi)$ is the set of subformulae of ψ.

Example 1. *Let* $\mathscr{X} = \{I, X, Y\}$, *let* $\tau_I = \tau_Y = \mathrm{Pr}$, $\tau_X = \mathrm{Pr} \to \mathrm{Pr}$, *let* $\Lambda(I) = \Lambda(X) = 1, \Lambda(Y) = 0$. *Let*

$$\begin{aligned} \delta(I) &= \emptyset & &\mapsto (X \, \neg P) \\ \delta(X) &= x \colon \mathrm{Pr} & &\mapsto (\Diamond x) \vee \Box Y \\ \delta(Y) &= \emptyset & &\mapsto (X \, Y) \end{aligned}$$

Let $\mathscr{A} = (\mathscr{X}, \Lambda, I, \delta, (\tau_X)_{X \in \mathscr{X}})$. *We will see a run of* $\mathscr{A}$ *in Example 2. This automaton corresponds to the* HFL-*formula (see Section 3 for a definition of* HFL*)* $\big(\mu X. \lambda x. \Diamond x \vee \Box \nu Y.(X \, Y)\big) \neg P$.

2.2 Acceptance

In the context of an APKA, an environment is either the empty environment e_0 or of the form $e = (f_1^X \mapsto (\psi_1, e_1), \ldots, f_{n_X}^X \mapsto (\psi_{n_X}, e_{n_X}), e')$ where the ψ_i are in $\mathscr{Q}$, i.e., subformulae of $\delta(X)$ for some X. We call e' the *parent environment* of e, and any environment reachable via the irreflexive, transitive closure of this relation a *predecessor* of e. A pair (ψ_j, e_j) is called a closure. We set $e(f_i^X) = (\psi_i, e_i)$. While the set of environments never appears explicitly, we tacitly assume that at any point during a run of an APKA,

the only environments in existence are e_0 and any environments the automaton has created so far. This also means that all environments have only finitely many predecessors.

A configuration in a run of the automaton over some tree $\mathscr{T}$ has the form $(t,(Q,e),e',\Gamma,\Delta)$, where t is subtree of $\mathscr{T}$, $Q \in \mathscr{Q}$ is a subformula of $\delta(X)$ for some X, e and e' are environments, Γ is a possibly empty stack of closures, and Δ is a finite sequence of priorities. In each configuration, if the type of the current closure is $\tau_1 \mapsto \cdots \tau_n \mapsto \mathrm{Pr}$ then there are n elements on the stack, and their types are, from bottom to top, $\tau_1, \ldots, \tau_n$. The latter invariant is by induction over the definition of the transition semantics.

A run over $\mathscr{T}, t_0$ is a possibly infinite sequence of configurations that begins with the initial configuration $(t_0,(X_{\mathrm{init}},e_0),e_0,\varepsilon,\varepsilon)$ and is produced by a two-player game between players $\exists$ and $\forall$. In each configuration, the next configuration is either produced deterministically, or one of the two players picks a successor. A run is accepting if $\exists$ wins the game according to a winning condition which we state later.

The transition semantics from $(t,(Q,e),e',\Gamma,\Delta)$ is as follows:

- If Q is $X \in \mathscr{X}$, the automaton transitions towards $\delta(X)$. The closures on the stack are, from bottom to top, the closures $(\psi_1,e_1''),\ldots,(\psi_{n_X},e_{n_X}'')$ of types $\tau_1^X,\ldots,\tau_{n_X}^X$. The automaton creates a new environment $e'' = (f_1^X \mapsto (\psi_1,e_1''),\ldots,f_{n_X}^X \mapsto (\psi_{n_X},e_{n_X}''),e')$, removes all these closures from the stack (which is now empty) and transitions to $(t,(\delta(X),e''),e'',\varepsilon,\Delta')$, where Δ' is Δ with the priority of Q appended.

- If Q is of the form $(\psi_1\,\psi_2)$, then the automaton pushes (ψ_2,e) on the stack and transitions to the configuration $(t,(\psi_1,e),e',\Gamma\cdot(\psi_2,e),e',\Delta)$.

- If Q is of the form f_j^X and not of type Pr, then the automaton transitions to $(t,e(f_j^X),e',\Gamma,\Delta)$.

- If Q is of the form f_j^X and of type Pr, and if $e(f_j^X) = (Q',e'')$ with $e' \neq e''$, then the automaton transitions to $(t,(Q,e),e'',\Gamma,\Delta')$ where e'' is the parent of e' and Δ' is Δ without the top element.

- If Q is of the form f_j^X, of type Pr, and if $e(f_j^X) = (Q',e'')$ with $e' = e''$, then the automaton transitions to $(t,e(f_j^X),e',\Gamma,\Delta)$.

- If Q is of the form $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$ then the automaton transitions to $(t,(\psi_1,e),e',\Gamma,\Delta)$, respectively $(t,(\psi_2,e),e',\Gamma,\Delta)$, depending on $\exists$'s, respectively $\forall$'s choice.

- If Q is of the form $\Diamond\varphi$ or $\Box\varphi$ then $\exists$, respectively $\forall$, chooses a successor $t' \in \{t0,t1\}$ and the automaton transitions towards $(t',(\varphi,e),e',\Gamma,\Delta)$.[1]

- If Q is of the form P or $\neg P$ then $\exists$ wins if $\mathscr{T},t \models Q$ and $\forall$ wins if $\mathscr{T},t \not\models Q$.

By induction, the transition relation alone determines the winner of all finite plays of the game. The winner of an infinite play is determined by the behavior of the priority stack (see the end of this subsection).

Note that, in a departure from the usual way the Krivine Abstract Machine works, we insist that the equivalent of lambda abstraction pop the entire stack via a string of lambda abstractions implicit in each $\delta(X)$. While this is no proper restriction in expressive power, it makes bookkeeping which fixpoint is currently being computed much easier (see Definition 4).

Before we formalize the winner of an infinite play of the acceptance game, we illustrate the transition semantics via an example.

Example 2. *Consider the infinite binary tree where only the first two levels are labeled by P. Since all subtrees on a level are isomorphic, we refer to the root as r and all subtrees of level i as t_i. An example run of the automaton $\mathscr{A}$ from Example 1 over this tree is depicted in Figure 2. This example is adapted from [10]. The highest priority that occurs infinitely often during the run is 1 and, hence, odd. However,*

---

[1] Over possibly finite trees, a player who is stuck loses the game.

Figure 2: Part of an example run of the APKA from Example 2.

$C_0 = (r,(I,e_0),e_0,\varepsilon,\varepsilon)$

$C_1 = (r,((X \neg P),e_0),e_0,\varepsilon,1)$

$C_2 = (r,(X,e_0),e_0,\neg P,1)$

$C_3 = (r,(((\Diamond x) \vee \Box Y),e_1),e_1,\varepsilon,11)$

$C_4 = (r,((\Box Y),e_1),e_1,\varepsilon,11)$

$C_5 = (t_1,(Y,e_1),e_1,\varepsilon,11)$

$C_6 = (t_1,((X Y),e_2),e_2,\varepsilon,110)$

$C_7 = (t_1,(X,e_2),e_2,(Y,e_2),110)$

$C_8 = (t_1,(((\Diamond x) \vee \Box Y),e_3),e_3,\varepsilon,1101)$

$C_9 = (t_1,((\Diamond x),e_3),e_3,\varepsilon,1101)$

$C_{10} = (t_2,(x,e_3),e_3,\varepsilon,1101)$

$C_{11} = (t_2,(x,e_3),e_2,\varepsilon,110)$

$C_{12} = (t_2,(Y,e_2),e_2,\varepsilon,1101)$

$C_{13} = (t_2,((X Y),e_4),e_4,\varepsilon,1100)$

$C_{14} = (t_2,(X,e_4),e_4,(Y,e_4),1100)$

$C_{15} = (t_2,(((\Diamond x) \vee \Box Y),e_5),e_5,\varepsilon,11001)$

$C_{16} = (t_2,((\Diamond x),e_5),e_5,\varepsilon,11001)$

$C_{17} = (t_3,(x,e_5),e_5,\varepsilon,1101)$

$C_{18} = (t_3,(x,e_5),e_4,\varepsilon,1100)$

$C_{19} = (t_3,(Y,e_4),e_4,\varepsilon,1100)$

$e_1 = (x \mapsto (\neg P,e_0),e_0)$

$e_4 = (\varepsilon,e_2)$

$e_2 = (\varepsilon,e_1)$

$e_5 = (x \mapsto (Y,e_4),e_4)$

$e_3 = (x \mapsto (Y,e_2),e_2)$

all these occurrences of 1 *except the first two are eventually removed from the priority stack and the remaining priorities are all* 0. *We will see later that this means that the automaton accepts.*

Definition 3. *Let $C_i = (t_i,(\psi_i,e_i),e_i',\Gamma_i,\Delta_i)$ be a configuration. If $\psi_i = X$ then we say that X occurs in C_i. An* occurrence *of a fixpoint variable is a configuration such that the variable occurs in that configuration. Moreover, $C_{i+1} = (t_i,(\delta(X),e_{i+1}),e_{i+1},\varepsilon,\Delta_i \cdot \Lambda(X))$ is such that e_{i+1} is new and there is a new priority on top of the priority stack. We say that e_{i+1} and this stack element are* tied to *this occurrence of X.*

The above means that there is a one-to-one correspondence between environments and occurrences of fixpoint variables: Reading a fixpoint variable X in a configuration entails creation of a new environment, sometimes denoted by e_X, and every environment e is created by an occurrence of a fixpoint variable X_e. Moreover, each priority on the priority stack is tied to a unique occurrence of a fixpoint and, hence, environment. The converse does not hold, since priorities can be removed from the priority stack. However, we will see below that environments that correspond to deleted priorities are not relevant to the remainder of a run.

Definition 4. *Let $C_i = (t_i,(\psi_i,e_i),e_i',\Gamma_i,\Delta_i)$ be a configuration. The automaton is said to be currently computing the fixpoint X if e_i' was created by an occurrence of X. It is currently computing the environment e_i', which is tied to an occurrence of X.*

Lemma 5. *Let $(C_i)_{i \in \mathbb{N}} = ((t_i,(\psi_i,e_i),e_i',\Gamma_i,\Delta_i)_{i \in \mathbb{N}}$ be a run. For some i, let $C_i = (t_i,(\psi_i,e_i),e_i',\Gamma_i,\Delta_i)$ be a configuration in that run.*

1. *e_i is either e_i' or a predecessor of e_i',*

2. *For e_i', all variable bindings point to closures (ψ,e') where e' is a predecessor of e_i' and the analogous property holds for all of e_i's predecessors,*

3. *all closures (ψ,e') on the stack are such that e' is either e_i or a predecessor of e_i,*

4. *The sequence of priorities on the priority stack is exactly the sequence of priorities tied to e_i' and the sequence of its predecessors.*

Proof. The proof is by induction over the sequence of configurations. After the initial state is expanded to its transition relation, the lemma holds. Item 2 needs to be verified only on environment creation, Item 1 only when the fixpoint currently being computed changes.

Now consider the form of the current closure (ψ_i, e_i) and assume that the lemma holds so far. Clearly, for modal and boolean operators there is nothing to prove.

If ψ_i is of the form $(\psi' \psi'')$, then ψ'' is put on the stack and, by assumption, is either from e_i or a predecessor environment, so again the new element conforms to Item 3.

If ψ_i is of the form X, then a new environment e_{i+1} is created and will be the new environment currently being computed. Moreover, the parent environment of e_{i+1} is e_i. This satisfies Item 1. Since all closures on the stack are from e_i or from predecessors of e_i, the new environment satisfies Item 2. Since the stack is empty, it fulfills the stack requirements. Moreover, a new priority is added to the priority stack. Since it is tied to the new environment, Item 4 continues to hold.

If ψ_i is a variable not of ground type, e_i switches to a predecessor and all items continue to hold. If ψ_i is a variable of ground type, the stack is empty and, hence, Item 3 is satisfied. There are two cases: Either $e_i(x) = (Q', e')$ with $e' = e_i'$, or, by Item 2, e' is a predecessor of e_i and, by Item 1, of e_i'. In the first case, the next closure will be (Q', e') computed in e', and there is nothing left to prove. In the second case, the automaton transitions towards $(t_i, (\psi_i, e_i), e_i'', \Gamma_i, \Delta_i')$, where e_i'' is the parent of e_i' and Δ_i' is Δ_i with the top priority removed. Hence, Item 4 is satisfied. Since e' is a predecessor of e_i', it is either equal to e_i'' or a predecessor of e_i'', so Item 1 is also satisfied. $\square$

From the definition of the transition relation, we can deduce that the environment which is currently being computed changes in two ways: By entering a new environment from its parent, which corresponds to environment creation, or by returning to the parent environment from an immediate successor environment. This means that, once an environment is left in favor of the parent environment, it will neve be returned to and the computation of its fixpoint is finished. Moreover, closures with this environment also never appear again. Hence, if such an environment is permanently left, we say that it is *being closed*. Formally, a closed environment is one such that a variable of ground type from this environment has been read or, equivalently, the automaton has reached a configuration $(t, (Q, e), e', \Gamma, \Delta)$ such that e' is the parent of the environment in question. Note that an environment is closed if and only if the corresponding priority has been removed from the priority stack.

Lemma 6. *Let e be an environment, let (ψ, e) be a closure of ground type for some configuration and let e be the environment currently being computed. As long as e stays the environment currently being computed, the type order of the current closure never properly decreases. If the computation changes from e to a proper successor and later returns to e for the next time, this happens in a ground-type proper subexpression of ψ.*

Proof. By the definition of the transition relation. The only transition that decreases the type order of the current closure is reading a fixpoint variable, which will change the environment currently being computed. Since (ψ, e) is of ground type, the stack must be empty. If the computation leaves e for a proper successor, this is through creation of a new environment or, equivalently, through reading a fixpoint variable. If the new environment binds a variable of ground type, the closure this variable points to must have been put on the stack between reading (ψ, e) and the environment's creation. Hence, it must be a proper subexpression of ψ. If the new environment does not bind a variable of ground type, the computation can not return to e. $\square$

Since $\delta(X)$ for each X has a finite syntax tree, repeated application of the previous lemma yields that the computation changes to any environment only a finite number of times. Otherwise we would obtain an infinitely descending sequence of subformulae of $\delta(X)$ where each subformula is an operand-type strict subformula of the previous.

It follows that each environment is either eventually closed, or eventually left permanently. Each environment appears as the environment currently being computed only finitely often. Moreover, each environment can only have finitely many direct successors because creation of a succesor of e during a configuration requires the previous configuration to be in e. This means that, during an infinite run, infinitely many environments will not be closed and the corresponding priorities will never be popped from the priority stack. We define that $\exists$ wins the acceptance game if the highest priority occurs infinitely often but is never popped from the stack is even.

More formally, consider a run $(C_i)_{i\in\mathbb{N}}$. Consider the subsequence of configurations $(C'_j)_{j\in J}$ such that $C'_j = (t_j, (\delta(X), e_j), e_j, \varepsilon, \Delta_j)$, i.e., a configuration such that e_j was created in this configuration, but such that there is no $i > j$ with a configuration $C_i = (t_i, (x, e_j), e_j, \Gamma_i, \Delta_i)$ with x of ground type, i.e., e_j is never closed. By the above considerations, J must be infinite. Then for all $n \geq j$, the priority stack Δ_j will be an initial segment of Δ_n. In particular, this holds for all $n \in J$. Hence, the set $(\Delta_j)_{j\in J}$ is is such that Δ_j is a prefix of $\Delta_{j'}$ if $j \leq j'$. We define that a play is accepting if the highest priority that occurs in the limit of this prefix-ordered chain is even. We say that an automaton $\mathscr{A}$ accepts a tree $\mathscr{T}$, and write $\mathscr{T} \models \mathscr{A}$, if and only if $\exists$ has a strategy such that the acceptance game generates an accepting run. Note that the above constitutes a *stair parity condition* in the sense that only those priorities contribute to the winning condition that are never removed from the priority stack.

Note that this is not the same as just taking the sequence of priorities occurring during the run: It is possible that a high priority occurs infinitely often during the run, but each occurrence is eventually removed from the priority stack. This occurs in Example 2 where priority 1 occurs infinitely often, but is always removed again from the priority stack a few configurations later.

Definition 7. *Two* APKA *are* equivalent *if and only if they accept the same trees.*

Observation 8. *For each* APKA $\mathscr{A}$ *there is an* APKA $\overline{\mathscr{A}}$ *over the same set of propositions such that for all trees, we have* $\mathscr{T} \models \mathscr{A}$ *if and only if* $\mathscr{T} \not\models \overline{\mathscr{A}}$.

The desired automaton is obtained by increasing the priorities of each state by one and replacing modal and boolean operators by their duals. A proof by induction over the structure of the acceptance game shows that a winning strategy for $\exists$ in the game for one automaton yields a winning strategy for $\forall$ in the other, and vice versa.

3 APKA **and** HFL

3.1 **Syntax of** HFL

In addition to the set $\mathscr{P}$ of atomic propositions, fix infinite sets of variables $\mathscr{V}$ disjoint from $\mathscr{F}$ and $\mathscr{Y}$ disjoint from $\mathscr{X}$ that denote variables bound by a λ-expression, respectively a fixpoint quantifier. Separating $\mathscr{V}$ and $\mathscr{Y}$ is usually not done for HFL, but facilitates technical exposition. Lower case letters $x, y, \ldots$ denote variables in $\mathscr{V}$, upper case letters $X, Y, \ldots$ those in $\mathscr{Y}$.

HFL-formulae φ are defined by the grammar

$$\varphi ::= P \mid \neg P \mid \Diamond\varphi \mid \Box\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid x \mid X \mid \lambda(x\colon \tau).\varphi \mid (\varphi\,\varphi) \mid \mu(X\colon \tau).\varphi \mid \nu(X\colon \tau).\varphi$$

Figure 3: Additional Typing Rules for HFL.

$$\frac{}{\Sigma,x\colon \tau \vdash x\colon \tau} \qquad \frac{}{\Sigma,X\colon \tau \vdash X\colon \tau} \qquad \frac{\Sigma,x\colon \tau \vdash \varphi\colon \tau'}{\Sigma \vdash \lambda(x\colon \tau).\varphi\colon \tau \to \tau'} \qquad \frac{\Sigma,X\colon \tau \vdash \varphi\colon \tau}{\Sigma \vdash \sigma(X\colon \tau).\varphi\colon \tau}$$

where $P \in \mathscr{P}$, $x \in \mathscr{V}$ and $X \in \mathscr{Y}$ and τ is a simple type. Note that negation is not present explicitly in the logic since it can be eliminated [11].

The binder $\lambda(x^v\colon \tau).\varphi$ binds x in φ, the binder $\sigma(X\colon \tau).\varphi$ with $\sigma \in \{\mu,\nu\}$ binds X in φ. Let $\mathrm{sub}(\varphi)$ be the set of subformulae of φ. An HFL-formula is *well-named* if there is, for each $X \in \mathscr{Y}$, at most one subformula of the form $\sigma(X\colon \tau).\psi$ and, for each $x \in \mathscr{V}$, at most one subformula of the form $\lambda(x\colon \tau).\psi$.

A variable from $\mathscr{V}$ or $\mathscr{Y}$ in a formula φ is *bound* if it is bound by a binder of the respective type, and *free* otherwise. A formula is called *closed* if it has no free variables and *open* otherwise. For a well-named formula φ and $X \in \mathscr{Y} \cap \mathrm{sub}(\varphi)$, define $\mathrm{fp}_\varphi(X)$ as the unique subformula ψ of φ such that $\psi = \sigma X.\psi'$ for $\sigma \in \{\mu,\nu\}$. We have a partial order $<_{\mathrm{fp}_\varphi}$ on the fixpoint variables of φ via $X <_{\mathrm{fp}_\varphi} Y$ if Y appears freely in $\mathrm{fp}_\varphi(X)$. We say that Y is *outermore* than X. A variable is outermost among a set of variables if it is maximal in this set with respect to $<_{\mathrm{fp}_\varphi}$.

We say that φ has type τ in a *context* Σ if $\Sigma \vdash \varphi\colon \tau$ can be derived via the typing rules in in Figures 1 and 3. Note that the rules concerning variables from $\mathscr{X}$ and $\mathscr{F}$ are not used. If $\Sigma \vdash \varphi\colon \tau$ for some Σ and τ then φ is *well-typed*. A closed formula is well typed if $\emptyset \vdash \varphi\colon \tau$. Typing judgments are unique if formulae are annotated with the correct types [14]. We usually omit the type annotations and tacitly assume that all formulae are well-typed and that the type of a formula can be derived from context.

3.2   Semantics of HFL

Fix a tree $\mathscr{T}$. The semantics of types are partially ordered sets defined inductively via $[\![\mathrm{Pr}]\!] = (2^T, \subseteq)$ and $[\![\tau \to \tau']\!] = ([\![\tau']\!]^{([\![\tau]\!])}, \sqsubseteq_{\tau \to \tau'})$, where $T = \{0,1\}^*$ and $[\![\tau']\!]^{([\![\tau]\!])}$ is the set of monotone functions from $[\![\tau]\!]$ to $[\![\tau']\!]$. Define the partial order $\sqsubseteq_{\tau \to \tau'}$ via pointwise comparison: For $f,g \in [\![\tau']\!]^{([\![\tau]\!])}$ let $f \sqsubseteq_{\tau \to \tau'} g$ if and only if $f(x) \sqsubseteq_{\tau'} g(x)$ for all $x \in [\![\tau]\!]$.

Note that $[\![\mathrm{Pr}]\!]$ is a boolean algebra and, hence, also a complete lattice. This makes $[\![\tau \to \tau']\!]$ also a complete lattice for all τ,τ'. Let $\bigsqcup_\tau M$ and $\bigsqcap_\tau M$ denote the join and meet, respectively, of the set $M \subseteq [\![\tau]\!]$, and let $\top_\tau$ and $\bot_\tau$ denote the maximal and minimal elements of $[\![\tau]\!]$.

Let $\Sigma = X_1\colon \tau_1,\ldots,X_n\colon \tau_n,x_1\colon \tau'_1,\ldots,x_m\colon \tau'_m$ be a context. An interpretation η is a partial map from the sets of variables $\mathscr{V}$ and $\mathscr{Y}$ such that $\eta(X_i) \in [\![\tau_i]\!]$ for all $i \leq n$ and $\eta(x_j) \in [\![\tau'_j]\!]$ for all $j \leq m$. Then $\eta[X \mapsto f]$ is the interpretation that maps X to f and agrees with η otherwise, similar for $\eta[x \mapsto f]$.

We define the semantics of HFL over $\mathscr{T}$ inductively as in Figure 4 (with dual cases left out for space considerations). For well-typed, well-named φ, we write $\mathscr{T},t \models_\eta \varphi$ if $s \in [\![\emptyset \vdash \varphi : \mathrm{Pr}]\!]_\eta$. We write $\mathscr{T},t \models \varphi$ if φ is closed and η is the empty interpretation. Two formulae are *equivalent*, written $\varphi \equiv \varphi'$, if $[\![\Sigma \vdash \varphi]\!]_\eta = [\![\Sigma \vdash \varphi']\!]_\eta$ for all η, Σ.

3.3   Translations between HFL and APKA

Lemma 9. *Let φ be an HFL-formula of order at most k. Then there is an APKA $\mathscr{A}_\varphi$ of order at most k such that, for all trees $\mathscr{T},t$, we have $\mathscr{T},t \models \varphi$ if and only if $\mathscr{A}_\varphi$ accepts $\mathscr{T},t$.*

Figure 4: Semantics of HFL.

$$\llbracket \Sigma \vdash P \colon \mathrm{Pr} \rrbracket_\eta = P^{\mathscr{T}}$$

$$\llbracket \Sigma \vdash \varphi \lor \psi \colon \mathrm{Pr} \rrbracket_\eta = \llbracket \Sigma \vdash \varphi \colon \mathrm{Pr} \rrbracket_\eta \cup \llbracket \Sigma \vdash \psi \colon \mathrm{Pr} \rrbracket_\eta$$

$$\llbracket \Sigma \vdash \Diamond \varphi \colon \mathrm{Pr} \rrbracket_\eta = \left\{ t \in T \colon t0 \in \llbracket \Sigma \vdash \varphi \colon \mathrm{Pr} \rrbracket_\eta \text{ or } t1 \in \llbracket \Sigma \vdash \varphi \colon \mathrm{Pr} \rrbracket_\eta \right\}$$

$$\llbracket \Sigma \vdash \lambda (x \colon \tau).\varphi \colon \tau \to \tau' \rrbracket_\eta = \{ f \in \llbracket \tau \to \tau' \rrbracket \colon \forall y \in \llbracket \tau \rrbracket.$$
$$f(y) = \llbracket \Sigma, x \colon \tau \vdash \varphi \colon \tau' \rrbracket_{\eta[x \mapsto y]} \}$$

$$\llbracket \Sigma \vdash X \colon \tau \rrbracket_\eta = \eta(X)$$

$$\llbracket \Sigma \vdash x \colon \tau \rrbracket_\eta = \eta(x)$$

$$\llbracket \Sigma \vdash \mu(X \colon \tau).\varphi \colon \tau \rrbracket_\eta = \bigsqcap \left\{ d \in \llbracket \tau \rrbracket \colon \llbracket \Sigma, (X \colon \tau) \vdash \varphi \colon \tau \rrbracket_{\eta[X \mapsto d]} \sqsubseteq_\tau d \right\}$$

$$\llbracket \Sigma \vdash (\varphi \, \psi) \colon \tau' \rrbracket_\eta = \llbracket \Sigma \vdash \varphi \colon \tau \to \tau' \rrbracket_\eta (\llbracket \Sigma \vdash \psi \colon \tau \rrbracket_\eta)$$

Proof. (Sketch) For space considerations, we only give a sketch of the proof. Let φ be a HFL-formula.

Since lambda abstraction is implicit for APKA and can only occur directly after a fixpoint, occurrences of lambda abstraction $\lambda f.\psi$ in φ that are not of the form $\sigma X.\lambda f_1.\ldots\lambda f_n.\psi$ need to be padded by vacuous fixpoints. If f is of type τ_1 and ψ is of type τ_2, replace $\lambda f.\psi$ by $\sigma X.\lambda f.\psi$, where X is of type $\tau_1 \to \tau_2$ and σ is chosen as convenient.

Next, free lambda variables are removed. For a subformula $\sigma X.\psi$ that contains a free variable f that is not a fixpoint, replace $\sigma X.\psi$ by $((\sigma X.\lambda f'.\psi[f'/f]) f)$ where f' is of the same type as f. This is organized such that fixpoints are translated before fixpoints in their subformulae, i.e., from top to bottom.

In a third step, any fixpoint of the form $\sigma X.\lambda f_1.\ldots\lambda f_n.\psi$ with ψ of type $\tau_1 \to \cdots \to \tau_m \to \mathrm{Pr}$ is changed to its η-long form, i.e., to $\sigma X.\lambda f_1.\ldots\lambda f_n.\lambda g_1.\ldots\lambda g_m.\psi'$ with $\psi' = ((\psi \, \tau_m) \cdots) \tau_1)$.

It is not hard to verify that neither of these steps changes semantics of the formula in question. Let φ' be the resulting HFL-formula and let $\mathscr{X}$ be the collection of fixpoint variables in φ'. Without loss of generality, φ' has the form $\sigma X_{\mathrm{init}}.\varphi''$ for some σ.

For each fixpoint X with defining formula $\sigma X.\lambda f_1.\ldots\lambda f_n.\psi$ set $\delta(X)$ to ψ where all occurrences of formula of the form $\sigma'X'.\psi'$ are replaced by X' and set τ_X as the type of X. Then the automaton $\mathscr{A} = (\mathscr{X}, \Lambda, X_{\mathrm{init}}, \delta, (\tau_X)_{X \in \mathscr{X}})$ with Λ chosen such that each fixpoint is labeled odd or even depending on parity, but not lower than any fixpoint in a subformula, is an APKA accepting the same trees as φ. $\quad\square$

Lemma 10. *Let $\mathscr{A}$ be an* APKA *of order at most k. Then there is an* HFL*-formula $\varphi_{\mathscr{A}}$ of order at most k such that, for all trees $\mathscr{T}, t$, we have $\mathscr{T}, t \models \varphi_{\mathscr{A}}$ if and only if $\mathscr{A}$ accepts $\mathscr{T}, t$.*

We skip the proof for space considerations. It rests on the idea that a fixpoint state X computes the formula $\sigma X.\lambda f_1^X.\ldots\lambda f_{n_X}^X.\delta(X)$ where σ is μ if $\Lambda(X)$ is odd and ν otherwise. However, the translation is subject to the same exponential blowup in size (but not in order) that occurs when translating ordinary parity automata into the modal μ-calculus. Moreover, further preprocessing is necessary because fixpoints can occur as operator-operand pair where the operator has a higher priority. In this case, a duplication of arguments is necessary to ensure proper precedence of fixpoints in the syntax tree. [2]

Corollary 11. *Emptiness of* APKA *is undecidable.*

---

[2] This idea is due to Naoki Kobayashi and Étienne Lozes.

Corollary 12. *For any* finite *tree $\mathcal{T}$ (or any finite Kripke structure), and any* APKA $\mathcal{A}$ *of order k, it is decidable in time k-fold exponential in the size of $\mathcal{T}$ whether $\mathcal{T} \models \mathcal{A}$.*

4   The Alternation Hierarchy for Alternating Parity Krivine Automata

4.1   Alternation Classes

We define the semantic alternation class via the least number of priorities of any equivalent automaton.

Definition 13. *We define the classes*

- Σ_n^{sem} *as the set of all* APKA *equivalent to one with at most n priorities such that the highest is even*

- Π_n^{sem} *as the set of all* APKA *equivalent to one with at most n priorities such that the highest is odd*

- $\Delta_n^{\text{sem}} = \Sigma_n^{\text{sem}} \cap \Pi_n^{\text{sem}}$.

Remark 14. *The following inclusions hold:*

$$\Sigma_n^{\text{sem}} \subseteq \Sigma_{n+1}^{\text{sem}} \qquad\qquad \Pi_n^{\text{sem}} \subseteq \Pi_{n+1}^{\text{sem}}$$
$$\Sigma_n^{\text{sem}} \subseteq \Pi_{n+1}^{\text{sem}} \qquad\qquad \Pi_n^{\text{sem}} \subseteq \Sigma_{n+1}^{\text{sem}}$$
$$\Sigma_n^{\text{sem}} \subseteq \Delta_{n+1}^{\text{sem}} \qquad\qquad \Pi_n^{\text{sem}} \subseteq \Delta_{n+1}^{\text{sem}}$$
$$\Delta_n^{\text{sem}} \subseteq \Delta_{n+1}^{\text{sem}}$$

Note that the alternation classes are independent of the order of an automaton. For a HFL-formula φ, we say that φ is in some alternation class if there is an equivalent APKA in that class.

Observation 15. *If $\mathcal{A} \in \Sigma_n^{\text{sem}}$ then $\overline{\mathcal{A}} \in \Pi_n^{\text{sem}}$, if $\mathcal{A}' \in \Pi_n^{\text{sem}}$ then $\overline{\mathcal{A}'} \in \Pi_n^{\text{sem}}$.*

4.2   Trees Encoding Acceptance Games

For each $n \geq 1$, define a set of propositions $\mathscr{P}_n$ as $\{D, C, V, T, F, F_0, \ldots, F_{n-1}\}$ as well as a set $\mathscr{P}_n'$ as $\{D, C, V, T, F, F_1, \ldots, F_n\}$.

Let $n \geq 1$. Consider a tree $\mathcal{T}$ and some APKA $\mathcal{A}$ with at most n priorities, over $\mathscr{P}_n$ or $\mathscr{P}_n'$ (depending on whether the highest priority is odd or even). We construct a tree $T(\mathcal{T}, \mathcal{A})$ over the same set of propositions which encodes the game tree of the acceptance game $G(\mathcal{T}, \mathcal{A})$ of $\mathcal{A}$ over $\mathcal{T}$. A state labeled by C signals that $\forall$ picks a successor configuration, a state labeled by D signals that $\exists$ picks a successor configuration, a state labeled by F_i signals that priority i is added to the priority stack in this configuration and a state labeled by V signals that the top priority is being removed. Configurations where the priority stack is not being manipulated and neither player picks a successor configuration are treated as if $\exists$ picks a successor, but both subtrees of $T(\mathcal{T}, \mathcal{A})$ are isomorphic.

The tree is generated inductively. Each position $(t, (Q, e), e', \Gamma, \Delta)$ in the acceptance game induces a subtree, with the root of the tree being generated by the initial position. At each vertex, exactly one proposition P from $\mathscr{P}_n$, respectively $\mathscr{P}_n'$ is true. We say that this vertex is labeled by P.

- The subtree induced by a position with Q of the form X is labeled $F_{\Lambda(X)}$. Both children are the subtree induced by $(t, (\delta(X), e''), e'', \Gamma', \Delta')$ where e'', Γ' and Δ' are as per the transition relation.

- The subtree induced by a position with Q of the form f_j^X is labeled V if both f_j^X is of type Pr and $e(f_j^X) = (Q', e'')$ with $e' \neq e''$. Otherwise, it is labeled D. Both children are the subtree induced by the successor configuration as per the transition relation.

Figure 5: Part of a $T(\mathscr{A},\mathscr{T})$ for $\mathscr{A}$ and $\mathscr{T}$ from Example 2. Omitted subtrees are isomorphic to their sibling if present or not shown for space considerations. C_i refers to the configuration from Figure 2 that induces the subtree and is not part of the label.
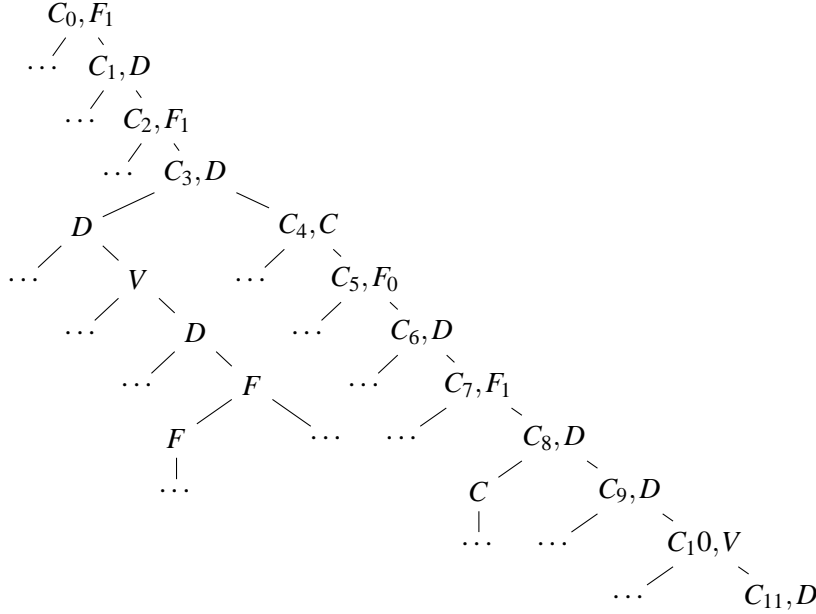
- The subtree induced by a position with Q of the form $(\psi_1 \psi_2)$ is labeled D. Both children are the subtree induced by $(t,(\psi_1,e),e',\Gamma(\psi_2,e),\Delta)$.

- The subtree induced by a position with Q of the form $\psi_1 \vee \psi_2$ is labeled D. The left subtree is the subtree induced by $(t,(\psi_1,e),e',\Gamma,\Delta)$, the right subtree is that induced by $(t,(\psi_2,e),e',\Gamma,\Delta)$.

- The subtree induced by a position with Q of the form $\psi_1 \wedge \psi_2$ is labeled C. The left subtree is the subtree induced by $(t,(\psi_1,e),e',\Gamma,\Delta)$, the right subtree is that induced by $(t,(\psi_2,e),e',\Gamma,\Delta)$.

- The subtree induced by a position with Q of the form $\Diamond\varphi$ is labeled D. The left subtree is the subtree induced by $(t0,(\varphi,e),e',\Gamma,\Delta)$, the right subtree is that induced by $(t1,(\varphi,e),e',\Gamma,\Delta)$.

- The subtree induced by a position with Q of the form $\Box\varphi$ is labeled C. The left subtree is the subtree induced by $(t0,(\varphi,e),e',\Gamma,\Delta)$, the right subtree is that induced by $(t1,(\varphi,e),e',\Gamma,\Delta)$.

- The subtree induced by a position with Q of the form P or $\neg P$ is labeled T if $\mathscr{T},s \models Q$ and F else. Both children are the subtree induced by $(t,(Q,e),e',\Gamma,\Delta)$ again.

It is easy to verify that this defines an infinite, fully binary tree. Figure 5 shows an example.

4.3 Hard Automata

We now construct APKA that are hard for their alternation classes. Following Arnold's [2] and Lange's [10] proofs, these automata accept trees encoding an acceptance game that is won by $\exists$, respectively $\forall$.

Consider the $\mathscr{P}_n$-, respectively $\mathscr{P}'_n$-APKA $\mathscr{A}_n^{\Sigma}$ and $\mathscr{A}_n^{\Pi}$ defined for each $n \geq 1$ as follows:

- The fixpoint states are $\{I,O,X_{n-1},\ldots,X_0\}$,

- the type of I is Pr, the type of the other states is $\mathrm{Pr} \to \mathrm{Pr}$,

- the initial state is I,

- $\Lambda(I) = \Lambda(O) = 0, \Lambda(X_i) = i$ for $\mathscr{A}_n^\Sigma$ and $\Lambda(I) = \Lambda(O) = 1$ and $\Lambda(X_i) = i+1$ for $\mathscr{A}_n^\Pi$,

- $\delta(I) = O\top, \delta(X_i) = x_0 \colon \mathrm{Pr} \mapsto (X_{i-1}x_0)$ for $i > 0, \delta(X_0) = x_0 \mapsto (Ox_0)$,

- $\delta(O) = x_0 \mapsto \neg F \wedge (T \vee \bigwedge \begin{pmatrix} (D) \to \Diamond(Ox_0) \\ (C) \to \Box(Ox_0) \\ (V) \to \Diamond(x_0) \\ (F_{n-1}) \to \Diamond(X_{n-1}(Ox_0)) \\ \cdots \\ (F_0) \to \Diamond(X_0(Ox_0)) \end{pmatrix})$.

Again, it is easy to verify that $\mathscr{A}_n^\Sigma \in \Sigma_n^{\mathrm{sem}}$ and that $\mathscr{A}_n^\Pi \in \Pi_n^{\mathrm{sem}}$.

These automata are equivalent to the HFL-formulae $(\sigma_{n-1}X_{n-1}.\lambda x_0((\cdots \sigma X_0.\lambda x_0.\psi)\cdots)x_0\top$ where the σ_i are μ, respectively ν depending on the alternation class, and $\psi = \delta(O)$.

Definition 16. *Consider a play of $\mathscr{A}_n^\Sigma$ over a $\mathscr{P}_n$-tree, respectively of $\mathscr{A}_n^\Pi$ over a $\mathscr{P}_n'$-tree generated from an acceptance game. A* round *in this play consists of a configuration where the current closure is O and all subsequent configurations until it is O again. An environment is* tied to a round *if it is created during that round.*

A round begins with the automaton in O. Unless the current tree node is labeled by F, $\forall$ chooses the right conjunct in $\delta(O)$, and $\exists$, unless the current state is labeled by T, chooses the right disjunct. $\forall$ then picks the conjunct indicated by the label of the current subtree in the big conjunction and $\exists$ picks the right part of the implication. Any different choice results in the player making that choice instantly loosing the game. One of the players is then in charge of picking a successor subtree. Depending on the conjunct picked by $\forall$, the game continues in a new instance of O, goes through $X_i, \ldots, X_0, O$ for some i or continues with the content of x_0. The latter will always lead to another instance of O, as we will see below. In either case, the game continues in the next round.

Observation 17. *Each round corresponds to exactly one configuration in the acceptance game of $\mathscr{A}$, namely that which induces the subtree in $T(\mathscr{T}, \mathscr{A})$ during the first configuration of the round. Furthermore, the current subtree in the game for $\mathscr{A}_n^\Sigma$ is labeled by C if and only if the configuration that induces it has a conjunction or a box as the top operator in the formula part of the current closure.*

Note that each configuration in a play for $\mathscr{A}_n^\Sigma$, respectively $\mathscr{A}_n^\Pi$ over suitable trees is part of exactly one round, with the exception of the first two configurations which have current closures (I) and $(O\top)$.

We call a round a *V-round* if $\forall$ picks the conjunct with V on the left of the implication, we call it an *F_k-round* if he picks the conjunct with F_k on the left of the implication and we call a round a *plain round* if he picks the conjuncts with C or D on the left of the implication. A round is *closed* if the environment tied to the single occurrence of O during that round is closed. A V-round is always closed immediately.

Lemma 18. *Consider a play of $\mathscr{A}_n^\Sigma$, respectively $\mathscr{A}_n^\Pi$ over a $\mathscr{P}_n$-tree, respectively $\mathscr{P}_n'$-tree generated from an acceptance game and let the automaton be at the start of some round, i.e., just before reading another occurrence of O. Let $(R_i)_{i \in I}$ be the sequence of unclosed rounds played so far, in order. Set $p^\Sigma(R) = 0$ if R is a plain round, set $p^\Pi(R) = 1$ if R is a plain round, set $p^\Sigma(R) = 0, k, \ldots, 1, 0$ if R is an F_k-round and set $p^\Pi(R) = 1, k+1, \ldots, 2, 1$ if R is a plain round. Then the priority stack of $\mathscr{A}_n^\Sigma$ from bottom to top is the concatenation of the $p^\Sigma(R_i)$ from first to last and the priority stack of $\mathscr{A}_n^\Pi$ from bottom to top is the concatenation of the $p^\Pi(R_i)$ from first to last.*

Moreover, all unclosed environments are tied to unclosed rounds. Tied to any plain round is a single environment for its ocurrence of O and it binds x_0 the last environment of the first unclosed round before. Tied to an F_k-round is a sequence of environments for the occurrences of $O, X_k, \ldots, X_0$. The environment for X_0 is the last environment, they all bind x_0 to x_0 of the previous environment except the environment for X_k which binds x_0 to (Ox_0) in the environment for O of its own round. Here, the initial unfolding for I is considered a dummy round.

Proof. The proof is by induction over the play. At the beginning of the very first round, the priority stack contains only the priority for I and $(\top, e_0)$ is on the stack. A plain round will consume the content from the stack, which is (x_0, e) of the previous round, or $(\top, e_0)$ for the very first round, and tie x_0 of its single ocurrence of O to it. Moreover, it will add 0 to the priority stack. An F_k-round R will also consume (x_0, e), respectively $(\top, e_0)$ from the stack and tie x_0 of the single ocurrence of O to it. During the round, the automaton will unfold X_k and tie (Ox_0) of that environment to X_k's x_0, then unfold $X_{k-1}, \ldots, X_0$ and create a chain of x_0 pointing to x_0 of the environment before. Moreover, it will put the sequence $p^{\Sigma}(R)$, respectively $p^{\Pi}(R)$ on the priority stack.

A V-round will put priority 0 on the stack, tie the x_0 of its single occurrence of O to x_0 of the previous unclosed round and then immediately read it. Consequently, all the environments of the previous unclosed round will be closed, including the ocurrrence of 0, and all the priorites tied to it will be popped. Notably, this will close all unclosed previous plain rounds until the next F_i-round, but nothing more. □

Lemma 19. *For all $\mathscr{P}_n$, respectively $\mathscr{P}'_n$-automata $\mathscr{A} \in \Sigma_n^{\text{sem}}$ and all infinite, fully binary $\mathscr{P}_n$-trees $\mathscr{T}$, we have that $\mathscr{T} \models \mathscr{A}$ if and only if $T(\mathscr{T}, \mathscr{A}) \models \mathscr{A}_n^{\Sigma}$ and for all $\mathscr{P}_n$, respectively $\mathscr{P}'_n$-automata $\mathscr{A}' \in \Pi_n^{\text{sem}}$, we have that $\mathscr{T} \models \mathscr{A}'$ if and only if $T(\mathscr{T}, \mathscr{A}') \models \mathscr{A}_n^{\Pi}$.*

Proof. We only show the case for $\mathscr{P}_n$ and we only show that $\exists$ has a winning strategy in the acceptance game for $\mathscr{A}_n^{\Sigma}$ over $T(\mathscr{T}, \mathscr{A})$ if she has one for $\mathscr{A}$ over $\mathscr{T}$, for a $\mathscr{P}_n$-automaton $\mathscr{A}$ in Σ_n^{sem} and $\mathscr{T}$ a $\mathscr{P}_n$-tree. The other cases are similar. Assume that $\exists$ has a winning strategy in the latter game.

The correspondence between rounds in the game for $T(\mathscr{T}, \mathscr{A})$ and configurations in the game for $\mathscr{A}$ suggests the following strategy for $\exists$ in the former game: Stay within subtrees that represent configurations that follow her winning strategy. Since the underlying game is assumed to be winning for $\exists$ and the root of $T(\mathscr{T}, \mathscr{A})$ represents such a configuration by assumption, she can maintain this invariant in any round where she picks the successor configuration. In rounds where $\forall$ picks the successor configuration, both of his choices must be winning for $\exists$ in the underlying game for $\mathscr{A}$ over $\mathscr{T}$ for otherwise the current configuration would not be winning for $\exists$. Clearly, following this strategy will guarantee that $\exists$ wins any finite play of the game for $\mathscr{A}_n^{\Sigma}$ by avoiding a node labeled F.

It remains to show that $\exists$ wins any infinite play when following the strategy above. This is because the sequence of unclosed nonplain rounds in the game for $\mathscr{A}_n^{\Sigma}$, and priority stack in the game for $\mathscr{A}$ correspond like this: If $(F_{k_i})_{i \in I}$ is the sequence of unclosed nonplain rounds, the $(k_i)_{i \in I}$ is the priority stack of $\mathscr{A}$. This follows from an induction over the two plays: Before the first round of the game for $T(\mathscr{T}, \mathscr{A})$, the sequence of unclosed rounds is empty, and so is the priority stack of the correspondig configuration of $\mathscr{A}$. Any plain round will add a 0, the least priority, to the priority stack of $\mathscr{A}_n^{\Sigma}$ and will not change the priority stack of $\mathscr{A}$. An F_k-round will add k to the priority stack of $\mathscr{A}$ and will add an unclosed F_k-round to the play of $\mathscr{A}_n^{\Sigma}$. A V-round will remove one priority k from the priority stack for $\mathscr{A}$ and will close a number of plain rounds and exactly one nonplain round in the game for $\mathscr{A}_n^{\Sigma}$. By the induction hypothesis, this is an F_k-round.

Hence, after both plays are finished, the highest priority to occur infinitely often on the stack for $\mathscr{A}$ is k if and only if there are infinitely many unclosed F_k-rounds, but only finitely many unclosed $F_{k'}$ rounds

for $k' > k$. It follows from Lemma 18 that the highest priority to occur infinitely often on the stack for $\mathscr{A}_n^\Sigma$ is k as well. Since $\exists$ wins the first game by assumption, that number must be even. $\qquad\square$

Lemma 20. *For each $n \geq 1$ and every $\mathscr{P}_n$-APKA $\mathscr{A} \in \Sigma_n^{\mathrm{sem}}$, there is a unique $\mathscr{T}^*$ such that $T(\mathscr{T}^*, \mathscr{A}) = \mathscr{T}^*$. For each $n \geq 1$ and every $\mathscr{P}_n$ APKA $\mathscr{A} \in \Pi_n^{\mathrm{sem}}$, there is a unique $\mathscr{T}^*$ such that $T(\mathscr{T}^*, \mathscr{A}) = \mathscr{T}^*$.*

Proof. The sets of all $\mathscr{P}_n$-trees, respectively the sets of all $\mathscr{P}_n'$-trees, form metric spaces via the metric $d(t, t') = 2^{-i}$, where i is the first level on which t and t' differ. It is well known that these spaces are complete [3]. Moreover, on all of these spaces, the mapping $f: \mathscr{T} \mapsto T(\mathscr{T}, \mathscr{A}_n^{\mathrm{sem}})$ is a contraction in the sense of the Banach Fixpoint Theorem since the game trees of two trees that differ at level i will coincide at least up to level $i + 1$. This is because the game with $\mathscr{A}_n^{\mathrm{sem}}$ transitions through I first and a full rotation through $\delta(O)$ for each level. Hence, by the Banach Fixpoint Theorem, f has a fixpoint $\mathscr{T}^*$. $\qquad\square$

Theorem 21. $\mathscr{A}_n^\Sigma \in \Sigma_n^{\mathrm{sem}} \setminus \Pi_n^{\mathrm{sem}}$ *and* $\mathscr{A}_n^\Pi \in \Pi_n^{\mathrm{sem}} \setminus \Sigma_n^{\mathrm{sem}}$.

Proof. For the sake of contradiction, assume that $\mathscr{A}_n^\Sigma \in \Pi_n^{\mathrm{sem}}$. Then $\mathscr{A}' = \overline{\mathscr{A}_n^\Sigma} \in \Sigma_n^{\mathrm{sem}}$. By Lemma 20, there is $\mathscr{T}^*$ such that $\underline{T(\mathscr{T}^*, \mathscr{A}')} = \mathscr{T}^*$. So by construction of $T(\mathscr{T}^*, \mathscr{A}')$, we have $\mathscr{T}^* \models \mathscr{A}_n^\Sigma$ iff $\mathscr{T}^* \models \mathscr{A}'$. But $\mathscr{A}' = \overline{\mathscr{A}_n^\Sigma}$, which is a contradiction. hence, $\mathscr{A}_n^\Sigma \notin \Pi_n^{\mathrm{sem}}$.

A similar proof works for the dual case. $\qquad\square$

Corollary 22. *For each n, $\Sigma_n^{\mathrm{sem}} \subsetneq \Sigma_{n+1}^{\mathrm{sem}}$ and $\Pi_n^{\mathrm{sem}} \subsetneq \Pi_{n+1}^{\mathrm{sem}}$.*

Proof. Since $\Pi_n^{\mathrm{sem}} \subseteq \Sigma_{n+1}^{\mathrm{sem}}$, non-strictness of $\Sigma_n^{\mathrm{sem}} \subseteq \Sigma_{n+1}^{\mathrm{sem}}$ would contradict the previous theorem. The same argument works for the dual case. $\qquad\square$

5   Discussion

It is a priori quite surprising that the order of an APKA or a HFL-formula is not of relevance when it comes to its alternation class. In particular, the automata $\mathscr{A}_n^\Sigma$ and $\mathscr{A}_n^\Pi$ that serve as example of automata that are hard for their respective classes are of order 1. This is surprising, since for the HFL-model-checking problem, which corresponds to acceptance for APKA, complexity is almost exclusively dictated by the order of a formula. We believe that this dichotomy stems from the way the transition relation for APKA is restricted to formulae of ground type. A state that would compute a higher-order function, say of type $(\mathrm{Pr} \to \mathrm{Pr}) \to (\mathrm{Pr} \to \mathrm{Pr})$ actually does not compute the full higher-order function, but its equivalent of type $(\mathrm{Pr} \to \mathrm{Pr}) \to \mathrm{Pr} \to \mathrm{Pr}$ at a fixed argument of type Pr. The first case requires computations over the full extent of a higher-order lattice, while in the second case it is sufficient to find an approximation that is good enough for the arguments in question.

Acknowledgements

References

[1] Rajeev Alur & P. Madhusudan (2004): *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*, ACM, pp. 202–211, doi:10.1145/1007352.1007390.

[2] André Arnold (1999): *The μ-calculus alternation-depth hierarchy is strict on binary trees*. ITA 33(4/5), pp. 329–340, doi:10.1051/ita:1999121.

[3] André Arnold & Maurice Nivat (1980): *The metric space of infinite trees. Algebraic and topological properties*. Fundam. Inform. 3(4), pp. 445–476.

[4] Julian C. Bradfield (1996): *The Modal mu-calculus Alternation Hierarchy is Strict*. In Ugo Montanari & Vladimiro Sassone, editors: *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings, Lecture Notes in Computer Science* 1119, Springer, pp. 233–246, doi:10.1007/3-540-61604-7_58.

[5] Florian Bruse (2014): *Alternating Parity Krivine Automata*. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger & Zoltán Ésik, editors: *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I, Lecture Notes in Computer Science* 8634, Springer, pp. 111–122, doi:10.1007/978-3-662-44522-8_10.

[6] E. Allen Emerson & Chin-Laung Lei (1986): *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*, IEEE Computer Society, pp. 267–278.

[7] Julian Gutierrez, Felix Klaedtke & Martin Lange (2014): *The μ-calculus alternation hierarchy collapses over structures with restricted connectivity*. Theor. Comput. Sci. 560, pp. 292–306, doi:10.1016/j.tcs.2014.03.027.

[8] Roope Kaivola (1995): *Axiomatising Linear Time Mu-calculus*. In Insup Lee & Scott A. Smolka, editors: *CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings, Lecture Notes in Computer Science* 962, Springer, pp. 423–437, doi:10.1007/3-540-60218-6_32.

[9] Jean-Louis Krivine (2007): *A call-by-name lambda-calculus machine*. Higher-Order and Symbolic Computation 20(3), pp. 199–207, doi:10.1007/s10990-007-9018-9.

[10] Martin Lange (2006): *The alternation hierarchy in fixpoint logic with chop is strict too*. Inf. Comput. 204(9), pp. 1346–1367, doi:10.1016/j.ic.2006.05.001.

[11] Étienne Lozes (2015): *A Type-Directed Negation Elimination*. In Ralph Matthes & Matteo Mio, editors: *Proceedings Tenth International Workshop on Fixed Points in Computer Science, FICS 2015, Berlin, Germany, September 11-12, 2015.*, EPTCS 191, pp. 132–142, doi:10.4204/EPTCS.191.12.

[12] Damian Niwinski (1997): *Fixed Point Characterization of Infinite Behavior of Finite-State Systems*. Theor. Comput. Sci. 189(1-2), pp. 1–69, doi:10.1016/S0304-3975(97)00039-X.

[13] Michael O. Rabin (1970): *Weakly Definable Relations and Special Automata*. In Yehoshua Bar-Hillel, editor: *Mathematical Logic and Foundations of Set Theory - Proceedings of an International Colloquium Held Under the Auspices of The Israel Academy of Sciences and Humanities, Studies in Logic and the Foundations of Mathematics* 59, Elsevier, pp. 1 – 23, doi:10.1016/S0049-237X(08)71929-3.

[14] Mahesh Viswanathan & Ramesh Viswanathan (2004): *A Higher Order Modal Fixed Point Logic*. In Philippa Gardner & Nobuko Yoshida, editors: *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings, Lecture Notes in Computer Science* 3170, Springer, pp. 512–528, doi:10.1007/978-3-540-28644-8_33.