On Computability of Logical Approaches to Branching-Time Property Verification of Programs

Takeshi Tsukada
The University of Tokyo
Japan
tsukada@kb.is.s.u-tokyo.ac.jp

Abstract

This paper studies the hardness of branching-time property verification of Turing-complete programming languages, as well as logical approaches to the verification problem. As these approaches reduce the verification problem to logical problems, e.g. the satisfiability problem of Horn clauses with certain extensions, it is natural to ask whether the logical problems are as hard as the verification problem or strictly harder. This paper reveals that logical problems used in most approaches are far more difficult than the verification problem; the only exception is the validity problem of first-order arithmetic with fixed-point operators. We also answers some other natural questions, for example, whether the extensions of Horn clauses are necessarily.

CCS Concepts: • **Theory of computation → Logic and verification**.

Keywords: program verification, branching-time property, computability, fixed-point logic, constrained Horn clause, analytical hierarchy

ACM Reference Format:
Takeshi Tsukada. 2020. On Computability of Logical Approaches to Branching-Time Property Verification of Programs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany.* ACM, New York, NY, USA, Article 1, 14 pages. https://doi.org/10.1145/3373718.3394766

1 Introduction

A fundamental question for a decision problem is how hard it is. Given a decision problem of interest, it is natural to ask whether the problem is decidable, whether there exists a polynomial-time algorithm and so on; they are about upper bounds of the hardness of the problem. Lower bounds would also be useful: if the problem is EXPSPACE-hard, one has to give up trying to reduce the problem to SAT.

This paper studies the hardness of program verification problems. The hardness of a verification problem depends on the programming language and the class of properties. Let us fix a sufficiently expressive programming language, say Rust, of which details are not significant in this paper.

Let us briefly review known results on the hardness of the verification problems for some classes.

The *halting problem*, of which the class of properties is singleton, is perhaps the most famous verification problem. This problem is Σ^0_1-complete. A slightly more general problem is the *reachability problem*, asking whether the evaluation of a given program reaches a certain state. It is also Σ^0_1-complete.

Another important class of properties is *safety*, saying that something bad will never happen. This is the dual of reachability and hence Π^0_1-complete. Because of its practical importance, many verification methods have been developed. Commonly used tools are decision procedures for the satisfiability problem of *Horn clauses* (see, e.g., [3] for the use of Horn clauses in this context), which is as hard as the verification problem (i.e. Π^0_1-complete).

Harel [9] studied the *fair termination* problem of nondeterministic programs. A program is fairly terminating if it has no infinite execution except for "unfair" ones. The fair termination problem is Π^1_1-complete, i.e. the hardest problem in those described by formulas in second-order arithmetic of the form $\forall X \subseteq \mathbb{N}.\varphi$ where φ has no second-order quantifier. The Π^1_1-completeness result can be extended to a fairly general class of *linear-time properties*: whether all execution paths of a given program satisfies a given property is Π^1_1, provided that the property is in a certain class containing all ω-regular word properties [9, 22].

This paper focuses on *branching-time property* verification of programs. So we are interested in the *tree* consisting of all execution paths of a given program, and the verification problem asks whether the execution tree satisfies a given ω-regular *tree* property. Let us write *Verif* for this verification problem. Several logical approaches have been proposed for the problem and sub-problems [1–4, 16, 21, 25].

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394766

The aim of this paper is to examine these approaches from the view point of computability. The logical problems used in these approaches are:

- the satisfiability problem of
  - constrained Horn clauses with some extensions, and
- the validity problems of
  - second-order arithmetic,
  - first-order arithmetic with fixed-point operators, and
  - higher-order arithmetic with fixed-point operators.

The results of this paper are summarised in Fig. 1 and Fig. 2.

We explain the meanings and consequences of the results, examining one-by-one problems listed above.

1.1 First-order fixed-point arithmetic

(First-order) fixed-point arithmetic [18] (or *μ-arithmetic*) is first-order arithmetic with least and greatest fixed-points, i.e. an arithmetic variant of the μ-calculus. Let *μ-arith* be the validity problem of μ-arithmetic.

Bradfield [4] proved, in effect, that $\mathit{Verif} \equiv_m \mu\text{-}\mathit{arith}$, i.e. the two problems are many-one reducible to each other.[1] This remarkable result, however, is rarely mentioned in the literature of program verification. In fact the above statement differs from the original statement, and this paper might be the first one writing Bradfield's result in the above form. (See Section 1.5 for more information.)

The characterisation of the verification problem in terms of μ-arithmetic has some interesting consequences. For example, $\mathit{Verif} \in \Delta^1_2$ since $\mu\text{-}\mathit{arith} \in \Delta^1_2$ (Proposition 7). In particular, the verification problem is far easier than the validity problem of second-order arithmetic.

1.2 Higher-Order Fixed-Point Arithmetic

Watanabe et al. [25] gave a reduction of the branching-time property verification problem to the validity problem of *higher-order fixed-point arithmetic* (*HFA* for short), extending the result in [16] for linear-time properties. HFA is an arithmetic variant of *higher-order modal fixed-point logic* [23], a higher-order variant of the modal μ-calculus. We write *HFA* (resp. HFA_n) for the validity problem of HFA (resp. the order-n fragment of HFA).

This paper proves two results:

$$\mathit{HFA} \in \Delta^1_2 \qquad \text{and} \qquad \forall n.\ \ \mathit{HFA}_n <_m \mathit{HFA}_{n+1}.$$

The first result may be surprising: despite the higher-order nature of HFA, it is far easier than second-order arithmetic. This is because HFA limits use of negation. In particular it prohibits negating a predicate variable, and thus every occurrence of a predicate variable is positive.

The second result also has a remarkable consequence:

$$(\text{order-}n \text{ program verification}) <_m \mathit{HFA}_n \qquad (n > 1).$$

[1]Actually polynomial-time reductions exist.

Note that order-n programs can be translated into order-1 programs by coding programs by natural numbers; this translation together with $\mathit{HFA}_1 <_m \mathit{HFA}_n$ gives the claim.

This is remarkable because the situation is quite different from the finite case. Here the finite case means the variant in which both the programming language and logic deal with only finite data (such as booleans), instead of natural numbers. In the finite case, Kobayashi *et al.* [12] gave polynomial-time reductions between the order-n program verification and the model-checking problem for order-n fixed-point logic. However the translation from the logical model-checking problem to the verification problem is somewhat unnatural; Walukiewicz asked whether there is a more natural translation [24, Section VIII]. This paper gives an evidence that this unnaturality is inevitable: there is no "natural" translation that is applicable to the infinite case as well.

1.3 Extensions of Constrained Horn Clauses

Beyene *et al.* [1] proposed an approach applicable to branching-time property verification, based on an extension of Horn clauses [2]. Let us consider four kinds of "clauses":

$$\begin{array}{llcl}
(\text{Base}) & \varphi \wedge H_1(\vec{x}_1) \wedge \cdots \wedge H_n(\vec{x}_n) & \to & G(\vec{y}) \\
(\vee) & \varphi \wedge H_1(\vec{x}_1) \wedge \cdots \wedge H_n(\vec{x}_n) & \to & G_1(\vec{y}_1) \vee G_2(\vec{y}_2) \\
(\exists) & \varphi \wedge H_1(\vec{x}_1) \wedge \cdots \wedge H_n(\vec{x}_n) & \to & \exists z. G(z, \vec{y}) \\
(\mathit{wf}) & & \mathit{wf}(H) &
\end{array}$$

Here $H_1, \dots, H_n, G, G_1, G_2$ are predicate variables, H is a predicate variable of arity 2, and a φ comes from a constraint language; in this paper, it is quantifier-free linear arithmetic. (Base) is the standard constrained Horn clause, and $(\vee)$ and $(\exists)$ are extensions allowing $\vee$ and $\exists$ at the head (i.e. the right-hand-side of $\to$); $\mathit{wf}(H)$ requires that the binary predicate H is well-founded, i.e. there is no infinite sequence $a_0 a_1 a_2 \dots$ such that all adjacent pairs of elements are related by H. Let us write $\mathit{CHC}[\exists, \mathit{wf}]$ for the satisfiability problem for finite sets of $(\exists)$- and (wf)-clauses in addition to (Base)-clauses. Beyene *et al.* [1] gave a reduction of the verification problem to $\mathit{CHC}[\exists, \mathit{wf}]$.

We show that $\mathit{CHC}[\exists, \mathit{wf}]$ is Σ^1_2-complete, which implies $\mathit{Verif} <_m \mathit{CHC}[\exists, \mathit{wf}]$. Then we sought sub-problems that is strictly easier than $\mathit{CHC}[\exists, \mathit{wf}]$ but expressive enough to deal with the verification problem. Unfortunately we cannot find such a sub-problem: all sub-problems that we checked are equiv-expressive to $\mathit{CHC}[\exists, \mathit{wf}]$ or too weak to handle Verif. Therefore $\mathit{CHC}[\exists, \mathit{wf}]$ seems an minimal extension for the purpose, although it is far harder than Verif.

1.4 Contributions

The contributions of this paper can be summarised as follows.

- We point out importance of first-order fixed-point arithmetic in branching-time verification. To the best of our knowledge, it is the unique logical problem that is as hard as branching-time property verification.

$$\begin{tikzcd}
 & & \vdots & & \\
CHC[\vee, wf], CHC[\exists, wf] \in & \Sigma^1_2 \arrow[ur, phantom, "\subsetneq" sloped] & & \Pi^1_2 \arrow[ul, phantom, "\supsetneq" sloped] & \\
Verif, \mu\text{-}arith, HFA \in & & \Delta^1_2 \arrow[ul, phantom, "\subsetneq" sloped] \arrow[ur, phantom, "\subsetneq" sloped] & & \\
CHC[\vee], CHC[\exists] \in & \Sigma^1_1 \arrow[ur, phantom, "\subsetneq" sloped] & & \Pi^1_1 \arrow[ul, phantom, "\supsetneq" sloped] & \ni CHC[wf] \\
 & & \Delta^1_1 \arrow[ul, phantom, "\subsetneq" sloped] \arrow[ur, phantom, "\subsetneq" sloped] & & \\
 & \Sigma^1_0 & = \Delta^1_0 = \arrow[u, phantom, "\subsetneq" sloped] & \Pi^1_0 &
\end{tikzcd}$$

Figure 1. Verification and logical problems placed in the analytical hierarchy (CHC = constrained Horn clauses, HFA = higher-order fixed-point arithmetic). For each problem, the lowest level in the hierarchy that contains the problem is shown. For example, *Verif* belongs to Δ^1_2 and higher levels, such as Σ^1_2 and Π^1_2, but not to Σ^1_1 nor Π^1_1.

$$Verif \quad \equiv_m \quad \mu\text{-}arith \quad \equiv_m \quad HFA_1 \quad <_m \quad HFA_2 \quad <_m \quad \dots \quad <_m \quad HFA_n \quad <_m \quad HFA_{n+1} \quad <_m \quad \dots \quad <_m HFA$$

Figure 2. Comparison of the problems in Δ^1_2. HFA_n is the restriction of HFA to order n.

- We prove basic results about the hardness of higher-order fixed-point arithmetic. Despite its higher-order nature, its validity problem is relatively easy among logical problems used in branching-time verification.
- We show that $CHC[\exists, wf]$ is harder than *Verif*, but it is a minimum extension to deal with *Verif*. We also make clear the hardness of its sub-problems.

At the end, we note that this theoretical analysis of logical problems does not immediately decide which approach is better, particularly from a practical perspective. Nevertheless, we think that the results motivate an extensive study of first-order fixed-point arithmetic in program verification.

1.5 Related Work

Bradfield [4] showed the strictness of the alternation hierarchy of the (propositional) modal μ-calculus. The idea is to transfer the alternation hierarchy of μ-arithmetic, which had been proved to be strict by Lubarsky [18]. To this end, he identified a class, say $\mathcal{K}$, of Kripke structures such that modal μ-calculus model-checking of $\mathcal{K}$ is equivalent to the validity problem *μ-arith* in a certain sense [4, Theorems 4 and 5]. Since the correspondence preserves the alternation depth of formulas, the equivalence result together with the strictness of the alternation hierarchy of μ-arithmetic implies the strictness of the alternation hierarchy of the modal μ-calculus.

The equivalence result of Bradfield [4] is relevant to our work since the class $\mathcal{K}$ consists of *effectively describable Kripke structures*, which can be identified with transition graphs of programs. Hence modal μ-calculus model-checking of an effectively describable Kripke structure can be seen as modal μ-calculus model-checking of a program, i.e. an instance of branching-time property verification of programs. His proofs are constructive, and it is easy to see that they induce (polynomial-time) reductions in both directions. In this way, Bradfield [4] in effect proved that *Verif* $\equiv_m$ *μ-arith*.

Bradfield [5] further studied the μ-arithmetic and gave a simple characterisation of the μ-arithmetic hierarchy in terms of the game quantifier.

Kobayashi *et al.* [13] gave a program verification method based on μ-arithmetic and proposed a way to solve the validity problem *μ-arith*.

Walukiewicz [24] studied a variant of λY-calculus having both least and greatest fixed points. A significant difference from higher-order fixed-point logics in [12, 16, 25] based on higher-order modal fixed-point logic [23] is that, in [24], a way of mixing least and greatest fixed points is controlled by a type system, an intersection-free variant of Kobayashi and Ong's system [14]. This type-based restriction simplifies a complicated winning criterion for higher-order fixed-point logic [6, 7] to the parity condition. The same technique is applicable to characterise a fragment HFA' of HFA that contains the image of the reductions of verification problems in [16, 25] and whose validity problem is as hard as the verification problem (i.e. *Verif* $\equiv_m HFA' <_m HFA$).

Unno *et al.* [21] and Nanjo *et al.* [19] discussed other logical approaches to temporal verification of programs. Their approaches reduce the verification problem to the validity problems of certain logics via type systems. The logics are quite expressive and the validity problems are far harder than *Verif*, despite that their methods focused on subproblems of *Verif*. Unno *et al.* [21] used second-order arithmetic; the logic in Nanjo *et al.* [19] is a kind of fixed-point logic, but it has quantifiers over infinite sequences by which second-order quantifiers can be coded.

2 Preliminaries

This section introduces basic notions and notations used in the sequel.

Given a set X, we write X^* for the set of all finite sequences over X. A sequence is written as $\langle x_1, \dots, x_n \rangle$. The set of natural numbers including 0 is written as $\mathbb{N}$. We write $\mathbb{N}_{>0}$ for the set of positive integers. We write $\mathcal{P}(X)$ for the powerset of the set X.

2.1 Computable Functions and Reductions

We assume the notion of computable functions on natural numbers; see, e.g., [11, 17].

A *decision problem* is a subset of natural numbers. We shall sometimes consider decision problems on a countable set X other than $\mathbb{N}$ via (implicit) coding. A is *many-one reducible* to B, written $A \leq_m B$, if there exists a total computable function $f : \mathbb{N} \to \mathbb{N}$ such that $n \in A \Leftrightarrow f(n) \in B$. If $A \leq_m B$ and $B \leq_m A$, then A and B are *many-one equivalent* (written $A \equiv_m B$). We write $A <_m B$ if $A \leq_m B$ but not $B \leq_m A$.

Remark 1. Many results of this paper are about many-one reducibility or irreducibility. One can strengthen some results by inspecting the proofs. Some reducibility results in fact give *polynomial-time* reductions, and some irreducibility results says that a problem A is even not *arithmetical* in another problem B (that means, there is no function $f : \mathbb{N} \to \mathbb{N}$ such that $n \in A \Leftrightarrow f(n) \in B$ and the graph of f can be defined by an arithmetic formula). □

We assume a computable pairing function $\langle -, - \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with computable projections $\pi_1, \pi_2 : \mathbb{N} \to \mathbb{N}$ such that $\pi_i(\langle n_1, n_2 \rangle) = n_i$, fixed in the sequel. The pairing function is assume to be bijective. The function can be extended to k-tuples by $\langle n_1 \rangle := n_1$ for $k = 1$ and $\langle n_1, \dots, n_k \rangle := \langle n_1, \langle n_2, \dots, n_k \rangle \rangle$ for $k > 2$. The corresponding projections are written as π^k_i, $1 \leq i \leq k$. The function $[-] : \mathbb{N}^+ \to \mathbb{N}$ from non-empty lists of natural numbers to natural numbers is defined by $[n_1, \dots, n_k] := \langle \langle n_1, \dots, n_k \rangle, k \rangle$.

2.2 Second-Order Arithmetic

We need some notions of *second-order arithmetic* because decision problems studied in this paper are far more difficult than first-order arithmetic.

Assume a countably infinite set of *term variables*, ranging over natural numbers. The set of *terms* is defined by the following grammar:

$$t, u \ ::= \ x \mid \mathsf{z} \mid \mathsf{S}(t) \mid t + u \mid t \times u.$$

We define $\underline{n}$ by $\underline{0} := \mathsf{z}$ and $\underline{n+1} := \mathsf{S}\, \underline{n}$.

Assume a countably infinite set of *predicate variables*, ranging over sets of natural numbers. The set of *formulas* of second-order arithmetic is given by

$$\begin{aligned} \varphi, \psi \ ::= \ & t = u \mid t \neq u \mid X\,t \mid \varphi \wedge \psi \mid \varphi \vee \psi \\ \mid \ & \forall x \leq t.\varphi \mid \exists x \leq t.\varphi \mid \forall x.\varphi \mid \exists x.\varphi \mid \forall X.\varphi \mid \exists X.\varphi. \end{aligned}$$

There are three kinds of quantifiers: *bounded quantifiers* ($\forall x \leq t$ and $\exists x \leq t$) on natural numbers, *first-order quantifiers* ($\forall x$ and $\exists x$) and *second-order quantifiers* ($\forall X$ and $\exists X$).[2] The meaning should be obvious. We sometimes write $\forall x \in \mathbb{N}.\varphi$ or $\forall x^{\mathbb{N}}.\varphi$ for $\forall x.\varphi$ and so on, emphasising the domain of the quantification.

The semantics of formulas is standard. A *valuation* ϱ maps term variables to natural numbers and predicate variables to sets of natural numbers. We write $\varrho[n/x]$ for the valuation defined by $\varrho[n/x](x) = n$ and $\varrho[n/x](y) = \varrho(y)$ (if $x \neq y$). We also write $[n/x]$ for $\varrho_0[n/x]$ where ϱ_0 maps every term variable to 0 and every predicate variable to the empty set. A pair (φ, ϱ) of a formula and a valuation determines a truth value $b \in \{\bot, \top\}$, which we write as $[\![\varphi]\!]_\varrho$; we omit its definition. We write $\varrho \models \varphi$ if $[\![\varphi]\!]_\varrho = \top$.

A formula is *bounded* or Δ^0_0 if it does not have first-order nor second-order quantifiers. A formula is Σ^0_n if it is of the form $\exists x_1.\forall x_2.\exists x_3.\dots.Q x_n.\varphi_0$ where φ_0 is bounded ($Q = \forall$ if n is even and otherwise $Q = \exists$). Similarly a formula of the form $\forall x_1.\exists x_2.\forall x_3.\dots.Q x_n.\varphi_0$ with bounded φ_0 is called a Π^0_n-formula. A formula is Σ^1_n if it is of the form $\exists X_1.\forall X_2.\dots Q X_n.\varphi_0$ where φ_0 is a formula with no second-order quantifier; Π^1_n-formulas are defined similarly.

Let $\varphi = \varphi(x)$ be a formula with a distinguished variable x. It defines a subset of natural numbers $\{\, n \in \mathbb{N} \mid [n/x] \models \varphi \,\}$. A subset $A \subseteq \mathbb{N}$ of natural numbers is *analytical* if it is defined by a formula of second-order arithmetic. For $i = 0, 1$ and $n \in \mathbb{N}$, a set is Σ^i_n (resp. Π^i_n) if it is defined by a Σ^i_n-formula (resp. Π^i_n-formula), and it is Δ^i_n if it is both Σ^i_n and Π^i_n. A set is Δ^0_1 if and only if it is *computable* (or *decidable*, *recursive*); a set is Σ^0_1 if and only if it is *computably enumerable* (or *recursively enumerable*).

The classes of Σ^1_n-, Π^1_n- and Δ^1_n-sets form a strict hierarchy, known as the *analytical hierarchy*:

$$\Sigma^1_n \supsetneq \Delta^1_n \subsetneq \Pi^1_n \qquad\qquad \Sigma^1_n \subsetneq \Delta^1_{n+1} \supsetneq \Pi^1_n$$

where Σ^1_n denotes the class of Σ^1_n-sets and so on. Any analytical set belongs to Σ^1_n for some n.

A set $A \subseteq \mathbb{N}$ is Σ^i_n*-hard* if $B \leq_m A$ for every Σ^i_n-set B. A Σ^i_n-hard set A is Σ^i_n*-complete* if it is Σ^i_n. Π^i_n*-hardness* and Π^i_n*-completeness* are defined similarly.

There is an important Π^1_1-complete set, closely related to verification. Let V be a set and $R \subseteq V \times V$ be a relation on V. An *infinite path* from $v_0 \in V$ is an infinite sequence $\langle v_0, v_1, \dots \rangle \in V^\omega$ such that $(v_i, v_{i+1}) \in R$ for every i. The relation R is *well-founded* from v_0 if R has no infinite path from v_0. Let $W\!F$ be the set of (code of) Σ^0_1-formulas $\varphi(x, y)$ such that $\{\, (n, m) \mid [n/x, m/y] \models \varphi \,\}$ is well-founded from 0.

[2] The domain of second-order quantifiers are sets in this paper, but replacing them with quantifiers over functions on natural numbers do not change anything in this paper. Note that function quantifiers are "definable" by using set quantifiers and first-order quantifiers.

By regarding φ as a description of a nondeterministic small-step reduction relation, well-foundedness corresponds to must-termination from the initial term represented by 0.

Proposition 2. *WF is Π^1_1-complete.*

Proof. This is a minor modification of a famous theorem (see, e.g., [11, Theorem XX, §16.4, p.396]). □

2.3 Games

Some proofs in this paper uses notions from game theory. This subsection briefly introduces notions used in the proofs.

Formally a *game* is a tuple $\mathcal{G} = (V_0, V_1, v_0, E, W)$ where:

- V_0 and V_1 are disjoint sets of 0-*nodes* and 1-*nodes*, respectively. Let $V := V_0 \cup V_1$.
- $v_0 \in V$ is an initial node.
- $E \subseteq V \times V$ is a set of (directed) *edges*.
- $W \subseteq V^\omega$ is a subset of infinite sequences over V, called the *winning condition*.

It is a two-player game, of which players are called 0 and 1. There is a token on a node, which is initially on v_0. In each turn, the token is moved a node connected by an edge from the current node. If the token is on an i-node, then the next node is chosen by Player i. If the play reaches a *dead-end*, i.e. a node with no outgoing edge, then the owner of the node loses. If the play continues indefinitely, the winner is determined by W: Player 0 wins when the play is in W, and Player 1 wins if it is not.

A *strategy* is a function $V^* \to V$. A pair (f_0, f_1) of strategies determines an infinite sequence $\langle f_0 \mid f_1 \rangle = v_0 v_1 \dots$ of nodes, which starts from the initial node v_0, by $v_{k+1} := f_i(v_0 \dots v_k)$ if $v_k \in V_i$. A strategy f_0 is a *winning strategy of Player* 0 if $\forall f_1.\ \langle f_0 \mid f_1 \rangle \in W \cup \bar{E}$, where

$$\bar{E} := \left\{ (v_i)_{i\in\omega} \;\middle|\; \exists k. \left(\begin{array}{c} \forall i < k.(v_i, v_{i+1}) \in E) \\ \qquad \wedge (v_k, v_{k+1}) \notin E \wedge v_k \in V_1 \end{array} \right) \right\}$$

is the set of infinite plays in which Player 1 first violates the rule E. A winning strategy of Player 1 can be defined similarly. Player i *wins* the game $\mathcal{G}$ if a winning strategy of Player i exists. A game $\mathcal{G}$ is *determined* if Player 0 or Player 1 wins.

The *dual game* is obtained by switching the roles of Player 0 and Player 1. For a game $\mathcal{G} = (V_0, V_1, v_0, E, W)$, its dual $\mathcal{G}^\perp$ is $(V_1, V_0, v_0, E, (V^\omega \setminus W))$. Obviously Player i wins $\mathcal{G}$ if and only if Player $(1 - i)$ wins $\mathcal{G}^\perp$.

A *parity game* is a game of which the winning condition is the *parity condition*. It is equipped with a function $\Omega : V \to \{0, \dots, k\}$, assigning the *priority* to each node. Given $v \in V^\omega$, let $\mathrm{Inf}_\Omega(v) \subseteq \{0, \dots, k\}$ be the set of numbers ℓ such that $\{i \mid \Omega(v_i) = \ell\}$ is infinite. Then $v \in V^\omega$ satisfies the *parity condition* if $\max\{\mathrm{Inf}_\Omega(v)\}$ is even. Every parity game is determined.

3 Branching-Time Property Verification

This section gives a formal definition of the branching-time property verification of programs and briefly reviews basic properties. The verification problem is intuitively defined as the modal μ-calculus model-checking problem of Kripke structures induced by programs (although the actual definition does not refer to any programming languages).

Assume a finite set of propositional variables PV. A *Kripke structure* $\mathcal{S}$ is a tuple (S, s_0, R, L) where S is a set of *states*, $s_0 \in S$ is an *initial state*, $R \subseteq S \times S$ is a *transition relation*, and $L : PV \to \mathcal{P}(S)$ is a *labelling function*.

Usually a verification problem is defined as a model-checking problem of the Kripke structure induced by a program. A program induces a Kripke structure, of which a state represents a state of a computer and the transition relation is given by stepwise execution of the program. Our definition relies on an abstract characterisation of induced Kripke structure.

Definition 3 (Effective Kripke structure)**.** An *effective Kripke structure* is a tuple $\vec{\varphi} = (\varphi_R(x, y), (\varphi_a(x))_{a\in PV})$ of Σ^0_1-formulas (with no free variables other than indicated ones). An effective Kripke structure represents a Kripke structure $\mathcal{S}_{\vec{\varphi}} = (\mathbb{N}, 0, R, L)$ where $(n, m) \in R \overset{\text{def}}{\Leftrightarrow} [n/x, m/y] \models \varphi_R$ and $n \in L(a) \overset{\text{def}}{\Leftrightarrow} [n/x] \models \varphi_a$. □

We assume that the reader is familiar with modal μ-calculus (see, e.g., [8] for an exposition).

The *branching-time property verification problem* (or simply *verification problem*), written *Verif*, is a variant of the modal μ-calculus model-checking problem taking an effective Kripke structure instead of a Kripke structure.

Remark 4. One can replace Σ^0_1-formulas in the definition of effective Kripke structures to primitive recursive formulas and to arithmetic formulas; the former is an restriction, and the latter is an extension. These changes do not affect the verification problem *Verif* in the sense that all variants are many-one reducible to each other. It is also equivalent to solving ω-regular games over infinite but computable graphs. This robustness justifies the definition. □

Let us informally discuss the relationship between *Verif* and branching-time property verification of programs.

Every Kripke structure induced by a program can be seen as an effective Kripke structure. The transition relation must be Σ^0_1 since stepwise execution must be done by an actual computer, and atomic propositions are usually decidable properties on states of a computer. So every pair of a program and a μ-calculus formula can be seen as an instance of *Verif*.

Conversely, a given pair of an effective Kripke structure $\vec{\varphi}$ and a modal μ-calculus formula ψ can be translated to a pair of a program $P_{\vec{\varphi}}$ and another formula ψ'. The program $P_{\vec{\varphi}}$ calculates φ_R and φ_a; then the transition relation R of the Kripke structure induced by $P_{\vec{\varphi}}$ can be divided into two parts $R = R_0 \cup R_1$, namely, R_0 that corresponds to φ_R and

R_1 that describes intermediate steps computing φ_R. The new formula ψ' behaves as the original formula ψ except that ψ' ignores the intermediate steps.

As mentioned in Sections 1.1 and 1.5, Bradfield proved that *Verif* is many-one equivalent to the validity problem of μ-arithmetic (that shall be formally defined in Section 4).

Theorem 5 (Bradfield [4])**.** *Verif* $\equiv_m$ *μ-arith.*

Actually the reductions runs in polynomial-time.

We shall give two results on computability of *Verif*. The first result is straightforward but proved for self-containdness.

Proposition 6. *Verif is Π^1_1-hard and Σ^1_1-hard. So Verif $\notin \Pi^1_1 \cup \Sigma^1_1$.*

Proof. For every Kripke structure $\mathcal{S} = (S, s_0, R, L)$, R is well-founded from s_0 if and only if $\mathcal{S} \models \mu X.\Box X$. This shows that *Verif* is Π^1_1-hard by Proposition 2. Since the negation of a given μ-calculus formula is computable, *Verif* is Σ^1_1-hard as well. By the strictness of the analytical hierarchy, no Σ^1_1-hard problem is Π^1_1; hence *Verif* is not Π^1_1. Similarly *Verif* $\notin \Sigma^1_1$. □

The second result is a corollary of Theorem 5 and Theorem 19 proved in the next section.

Proposition 7. *Verif* $\in \Delta^1_2$.

Remark 8. Lubarsky [18] proved that the sets of natural numbers definable by μ-arithmetic formulas are Δ^1_2-sets. We note that μ-*arith* $\in \Delta^1_2$ is stronger than Lubarsky's result. □

4 Higher-Order Fixed-Point Arithmetic

Higher-order modal fixed-point logic [23] is an extension of the modal μ-calculus by higher-order features. Its arithmetic version, which we call *higher-order fixed-point arithmetic* (*HFA* for short), has been studied recently in Kobayashi *et al.* [16] and Watanabe *et al.* [25] and applied to temporal verification of higher-order programs.

This section proves two results:

$$\mathit{HFA} \in \Delta^1_2 \qquad \text{and} \qquad \forall n.\ \ \mathit{HFA}_n <_m \mathit{HFA}_{n+1}.$$

Here *HFA* (resp. HFA_n) is the validity problem of HFA (resp. the order-n fragment of HFA).

This section is organised as follows. The logic is defined in Section 4.1. We prove the former result in Section 4.2 and the latter result in Section 4.3.

4.1 Definition of Higher-Order Fixed-Point Arithmetic

Higher-order fixed-point arithmetic is a simply-typed calculus. The syntax of *types* is given by:

$$\begin{array}{lccl} \textit{complete types} & \tau, \sigma & ::= & \mathtt{Prop} \mid \vartheta \to \tau \\ \textit{argument types} & \vartheta & ::= & \tau \mid \mathtt{Nat}. \end{array}$$

Note that $\mathtt{Nat}$ cannot be a result of a function, and this is the only difference between $\mathtt{Nat}$ and other types. The *order* of a type is inductively defined as follows:

$$\mathit{order}(\mathtt{Prop}) := 1 \qquad \mathit{order}(\mathtt{Nat}) := 0$$
$$\mathit{order}(\vartheta \to \tau) := \max(\mathit{order}(\vartheta) + 1, \mathit{order}(\tau)).$$

The syntax of *terms* and *formulas* is given by:

$$\begin{array}{rcl} t, u & ::= & x \mid \mathsf{z} \mid \mathsf{S}(t) \mid t + u \mid t \times u \\ \varphi, \psi & ::= & t = u \mid t \neq u \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \forall x^{\mathtt{Nat}}.\varphi \mid \exists x^{\mathtt{Nat}}.\varphi \\ & \mid & x \mid \lambda x^{\vartheta}.\varphi \mid \varphi\, \psi \mid \varphi\, t \\ & \mid & \mu x^{\tau}.\varphi \mid \nu x^{\tau}.\varphi. \end{array}$$

The connectives from first-order arithmetic are listed in the first line. The second line consists of constructors of the simply-typed lambda calculus; since this syntax distinguishes terms from formulas, we need two kinds of applications. The least and greatest fixed-point operators are listed in the third line. We shall often omit the type annotations.

Remark 9. One can remove summation $t + u$, multiplication $t \times u$ and inequality $t \neq u$ without sacrificing the expressiveness; see Example 10. □

Free and bound variables are defined in the standard way. The binders are $\forall x$, $\exists x$, λx, μx and νx. We shall identify α-equivalent formulas.

Only well-typed terms and formulas are of interest. A *type environment* Γ is a finite list of type bindings of the form $x : \vartheta$ in which each variable occur at most once. A *type judgement* is of the form $\Gamma \vdash \varphi : \tau$ or $\Gamma \vdash t : \mathtt{Nat}$. We show examples of typing rules:

$$\frac{\Gamma \vdash \varphi : \mathtt{Prop} \quad \Gamma \vdash \psi : \mathtt{Prop}}{\Gamma \vdash \varphi \wedge \psi : \mathtt{Prop}}$$

$$\frac{\Gamma, x : \mathtt{Nat} \vdash \varphi : \mathtt{Prop}}{\Gamma \vdash \forall x^{\mathtt{Nat}}.\varphi : \mathtt{Prop}} \qquad \frac{\Gamma, x : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu x^{\tau}.\varphi : \tau}.$$

A *typed formula* is a formula φ with a derivation $\Gamma \vdash \varphi : \tau$. All formulas appearing in the sequel are typed, and hence we simply call them formulas. A typed formula $\Gamma \vdash \varphi : \tau$ is *closed* if the environment Γ is empty. A *sentence* is a closed formula of type $\mathtt{Prop}$.

We define the semantics of types and formulas.

Each type denotes a poset.

- $[\![\mathtt{Prop}]\!] := \Omega$, the poset of truth values, i.e. $\Omega = \{\bot, \top\}$ ordered by $\bot <_{\mathtt{Prop}} \top$.
- $[\![\mathtt{Nat}]\!] := \mathbb{N}$ with the discrete order, i.e. $n \leq_{\mathtt{Nat}} m$ if and only if $n = m$.
- $[\![\vartheta \to \tau]\!]$ is the set of all *monotone* functions from $[\![\vartheta]\!]$ to $[\![\tau]\!]$ with the point-wise ordering, i.e. $f \leq_{\vartheta \to \tau} g$ if and only if $\forall x \in [\![\vartheta]\!]. f(x) \leq_\tau g(x)$.

Note that the denotation $[\![\tau]\!]$ of a complete type τ is a complete lattice.

For a type environment $\Gamma = (x_1\colon \vartheta_1, \dots, x_n\colon \vartheta_n)$, its interpretation is the set of mappings ϱ with domain $\{x_1, \dots, x_n\}$ such that $\varrho(x_i) \in [\![\vartheta_i]\!]$ for every i. They are ordered by the point-wise ordering. An element of $[\![\Gamma]\!]$ is called a *valuation*.

A formula $\Gamma \vdash \varphi\colon \tau$ is interpreted as a function from $[\![\Gamma]\!]$ to $[\![\tau]\!]$, of which the value at $\varrho \in [\![\Gamma]\!]$ is written as $[\![\varphi]\!]_\varrho$. The semantics is defined by induction on φ. The most important rules are

$$[\![\mu x^\tau.\varphi]\!]_\varrho := \bigwedge \{\, v \in [\![\tau]\!] \mid [\![\varphi]\!]_{\varrho[x\mapsto v]} \leq_\tau v \,\}$$
$$[\![\nu x^\tau.\varphi]\!]_\varrho := \bigvee \{\, v \in [\![\tau]\!] \mid v \leq [\![\varphi]\!]_{\varrho[x\mapsto v]} \,\}.$$

Here $\bigvee$ and $\bigwedge$ are the join and meet of sets, which exist since $[\![\tau]\!]$ is complete. Knaster-Tarski fixed-point theorem ensures that $[\![\mu x^\tau.\varphi]\!]_\varrho$ and $[\![\nu x^\tau.\varphi]\!]_\varrho$ are indeed the least and greatest fixed points of the mapping $v \mapsto [\![\varphi]\!]_{\varrho[x\mapsto v]}$ since the mapping is monotone.

Another definition of the least fixed-point would also be useful. Let $f\colon A \longrightarrow A$ be a monotone function on a complete lattice A. For each ordinal γ, we define $f^\gamma(x)$ by $f^0(x) := x$, $f^{\gamma+1}(x) = f(f^\gamma(x))$ and, for a limit ordinal γ,

$$f^\gamma(x) := \bigvee_{\gamma' < \gamma} f^{\gamma'}(x).$$

Then $(f^\gamma(\bot))_\gamma$ is an increasing sequence that is constant for sufficiently large ordinals. This constant is the least fixed-point of f.

Example 10. Let *plus* be a formula

$\mu plus.\lambda a.\lambda b.\lambda c.$
$(b = \mathsf{z} \wedge a = c) \vee (\exists b'c'.b = \mathsf{S}(b') \wedge c = \mathsf{S}(c') \wedge plus\, a\, b'\, c')$.

This formula can be seen as the definition of summation since $[\![plus\, s\, t\, u]\!]_\varrho = [\![s + t = u]\!]_\varrho$. Let *lt* be a formula

$$\mu lt.\lambda a.\lambda b.(\mathsf{S}(a) = b) \vee lt\,(\mathsf{S}(a))\, b.$$

This formula is equivalent to $<$ in an appropriate sense. Then $s \neq t$ can be defined as $lt\, s\, t \vee lt\, t\, s$.[3] □

Remark 11. *HFA*$_n$-formulas are *not* closed under negation; free higher-order variables are problematic. However, restricted to "first-order predicates", i.e. formulas of the form $x_1\colon \mathtt{Nat}, \dots, x_n\colon \mathtt{Nat} \vdash \varphi : \mathtt{Prop}$, the negation is a definable operation. It is obtained by simply replacing each logical connective to its De Morgan daul:

$$\vee \leftrightsquigarrow \wedge, \qquad = \leftrightsquigarrow \neq, \qquad \exists \leftrightsquigarrow \forall, \quad \text{and} \quad \nu \leftrightsquigarrow \mu.$$

We shall write $\neg\varphi$ for the negation of φ. □

Given a typed formula $\Gamma \vdash \varphi\colon \mathtt{Prop}$ and a valuation $\varrho \in [\![\Gamma]\!]$, we write $\varrho \models \varphi$ if and only if $[\![\varphi]\!]_\varrho = \top$.

Let *HFA* be the set of (code of) true sentences φ, i.e.,

$$HFA := \{\, \llcorner\varphi\lrcorner \mid \emptyset \models \varphi \,\}.$$

We write *HFA*$_n$, $n \geq 1$, for the restriction of *HFA* to order-n sentences (i.e. the set of (code of) true order-n sentences, where the *order* of a formula is the maximum order of types that appear in the typing derivation). We often say $\varphi \in HFA$ to mean $\llcorner\varphi\lrcorner \in HFA$.

[3] The author is grateful to Mayuko Kori who pointed out this encoding of $\neq$.

4.2 Operational Game Semantics

This subsection proves that *HFA* is in Δ^1_2. The basic idea is to express the evaluation of a given formula in terms a game of which the underlying graph represents the small-step operational semantics. A similar construction is well-known for modal μ-calculus as well as (first-order) μ-arithmetic. A significant difference from the first-order cases can be found in the winning condition: the game of this subsection is no longer a parity game, but a game with an uncommon winning criterion as in [6, 7, 16, 25].[4]

The "operational semantics" of formulas is defined as follows. We annotate a label to each fixed-point operators in a formula, in order to track the caller-callee relation: the label ℓ of a fixed-point operator $\mu^\ell x.\varphi$ indicates the name of the *parent*. The set of labels can be arbitrary infinite sets, and we use the set of natural numbers. An *configuration* is of the form $\langle\varphi\rangle^\ell_T$, where φ is a sentence, $T \subseteq \mathbb{N} \times \{\mu, \nu\} \times \mathbb{N}$ is a finite edge-labelled tree, and ℓ is the maximum of the labels that have been used.

$$\begin{aligned}
\langle(\mu^\ell x.\varphi)\,\vec{\psi}\rangle^{\ell'}_T &\longrightarrow \langle(\varphi\{(\mu^{\ell'+1}x.\varphi)/x\})\,\vec{\psi}\rangle^{\ell'+1}_{T\cup(\ell,\mu,\ell'+1)} \\
\langle(\nu^\ell x.\varphi)\,\vec{\psi}\rangle^{\ell'}_T &\longrightarrow \langle(\varphi\{(\nu^{\ell'+1}x.\varphi)/x\})\,\vec{\psi}\rangle^{\ell'+1}_{T\cup(\ell,\nu,\ell'+1)} \\
\langle(\lambda x.\varphi)\,\varphi'\,\vec{\psi}\rangle^\ell_T &\longrightarrow \langle(\varphi\{\varphi'/x\})\,\vec{\psi}\rangle^\ell_T \\
\langle\varphi_1 \wedge \varphi_2\rangle^\ell_T &\longrightarrow \langle\varphi_i\rangle^\ell_T, && i = 1, 2 \\
\langle\varphi_1 \vee \varphi_2\rangle^\ell_T &\longrightarrow \langle\varphi_i\rangle^\ell_T, && i = 1, 2 \\
\langle\forall x.\varphi\rangle^\ell_T &\longrightarrow \langle\varphi[n/x]\rangle^\ell_T, && n \in \mathbb{N}, \\
\langle\exists x.\varphi\rangle^\ell_T &\longrightarrow \langle\varphi[n/x]\rangle^\ell_T, && n \in \mathbb{N},
\end{aligned}$$

where unlabelled fixed-point operators $\mu X.\varphi$ and $\nu X.\varphi'$ are regarded as those labelled by 0. Ignoring ℓ and T, most rules are the standard call-by-name operational semantics. The last two rules can be understood as nondeterministic choice of a natural number.

We explain the key rule, namely the first and second rules. Consider the case that the head is $\mu^\ell x.\varphi$, where ℓ is the name of the parent of this fixed-point operator. One first generates a fresh label $\ell' + 1$, which is the name of this μ, and then records the parent-children correspondence $(\ell, \mu, \ell' + 1)$. After that, the fixed-point operator is expanded; newly created μ's are labelled by $\ell' + 1$, the name of the current μ.

Definition 12. Assume an infinite reduction sequence

$$\langle\varphi\rangle^0_\emptyset = \langle\varphi_0\rangle^{\ell_0}_{T_0} \longrightarrow \langle\psi_1\rangle^{\ell_1}_{T_1} \longrightarrow \cdots \longrightarrow \langle\psi_n\rangle^{\ell_n}_{T_n} \longrightarrow \cdots.$$

[4] The introduction of an uncommon winning criterion is inevitable. Since *Verif* $\equiv_m$ *HFA*$_1$ $<_m$ *HFA*$_n$ for $n > 1$, parity games on computable graphs (which are instances of *Verif*) are too weak to precisely capture the semantics of order-n formulas ($n > 1$).

The edge-labelled tree $T := \bigcup_{i \in \omega} T_i$ is called the *call tree*. For every path of T, all edges have the same label; a path is a μ-path (resp. ν-path) if the label on edges is μ (resp. ν).

Example 13. Let φ and ψ be HFA formulas given by

$$\begin{aligned}\varphi &:= \mu F.\lambda g.\lambda x.g\,x\,(F\,g\,(\mathsf{S}(x)))\\ \psi &:= \mu G.\lambda y.\lambda p.(y = \mathsf{z} \wedge p) \vee (\exists y'.y = \mathsf{S}(y') \wedge G\,y'\,p)\end{aligned}$$

where $F\colon (\mathtt{Nat} \to \mathtt{Prop} \to \mathtt{Prop}) \to \mathtt{Nat} \to \mathtt{Prop}$ and $G\colon \mathtt{Nat} \to \mathtt{Prop} \to \mathtt{Prop}$. It is not difficult to see that $\varphi\,\psi\,n$ is valid for every closed term $n\colon \mathtt{Nat}$.

Consider the following strategies of Player 0 and Player 1 for the game for $\varphi\,\psi\,\mathsf{z}$:

- Player 0 chooses the left-branch of $\vee$ if the value assigned to y is z; otherwise Player 0 chooses the right branch and set y' to $y - 1$.
- Player 1 always chooses the right-branch of $\wedge$.

The resulting infinite play is

$$\begin{aligned}\langle \varphi\,\psi\,\underline{0}\rangle^0_\emptyset &\longrightarrow^* \langle \psi\,\underline{0}\,(\varphi^{(1)}\,\psi\,\underline{1})\rangle^1_{T_1}\\ &\longrightarrow^* \langle (\underline{0} = \underline{0} \wedge (\varphi^{(1)}\,\psi\,\underline{1})) \vee (\cdots \psi^{(2)} \cdots)\rangle^2_{T_2}\\ &\longrightarrow^* \langle \varphi^{(1)}\,\psi\,\underline{1}\rangle^2_{T_2}\\ &\longrightarrow^* \langle \psi\,\underline{1}\,(\varphi^{(3)}\,\psi\,\underline{2})\rangle^3_{T_3}\\ &\longrightarrow^* \langle (\cdots) \vee (\exists y'.\underline{1} = \mathsf{S}(y') \wedge \psi^{(4)}\,y'\,(\varphi^{(3)}\,\psi\,\underline{2}))\rangle^4_{T_4}\\ &\longrightarrow^* \langle \psi^{(4)}\,\underline{0}\,(\varphi^{(3)}\,\psi\,\underline{2})\rangle^4_{T_4}\\ &\longrightarrow^* \langle (\underline{0} = \underline{0} \wedge (\varphi^{(3)}\,\psi\,\underline{2})) \vee (\cdots \psi^{(5)} \cdots)\rangle^5_{T_5}\\ &\longrightarrow^* \langle \varphi^{(3)}\,\psi\,\underline{2}\rangle^5_{T_5}\\ &\longrightarrow^* \langle \psi\,\underline{2}\,(\varphi^{(6)}\,\psi\,\underline{3})\rangle^6_{T_6}\\ &\longrightarrow \ldots\end{aligned}$$

where

$$\varphi^{(i)} := \nu^{(i)}F.\ldots \quad \text{and} \quad \psi^{(i)} := \mu^{(i)}G.\ldots$$

are labelled versions of φ and ψ. Occurrences of φ generated by the first expansion of νF are labelled by (1), and those by the second expansion are (3). The unique occurrence of ψ generated by the first expansion of μG is labelled by (2) and immediately discarded; the labelled formula $\psi^{(4)}$ is obtained by the second expansion of μG, one of which is further expanded, generating $\psi^{(5)}$. The tree T_6 is

$$\begin{tikzcd} & 1 \arrow[r] & 3 \arrow[r] & 6 & . \\ 0 \arrow[ur] \arrow[r] \arrow[dr] & 2 & & & \\ & 4 \arrow[r] & 5 & & \end{tikzcd}$$

The associated call tree $T = \bigcup_i T_i$ has a unique infinite path starting from 1. □

The following is the key lemma. This is essentially the same as [6, Lemma 6] and [15, Lemma 26, Appendix E.2], and we omit the proof.

Lemma 14. *For every HFA sentence $\vdash \varphi : \mathtt{Prop}$ and every infinite reduction sequence starting from $\langle\varphi\rangle_\emptyset$, the associated call tree has a unique infinite path.*

Definition 15. The *operational game* $\mathcal{G}(\varphi)$ of a given HFA sentence $\vdash \varphi : \mathtt{Prop}$ is defined by the following data:

- the game graph is defined by the reduction relation $\longrightarrow$; the initial node is $\langle\varphi\rangle^0_\emptyset$;
- the owner of $\langle\varphi_1 \vee \varphi_2\rangle^\ell_T$ and $\langle\exists x.\varphi\rangle^\ell_T$ is Player 0; the owner of $\langle\varphi_1 \wedge \varphi_2\rangle^\ell_T$ and $\langle\forall x.\varphi\rangle^\ell_T$ is Player 1; the owner of true (resp. false) atomic formulas is Player 1 (resp. 0); the owner of other nodes does not matter (because other nodes have unique successor) but for definiteness we set the owner Player 0;
- an infinite play is winning if the unique infinite path of the associated call tree is a ν-path.

We prove the correctness of the game semantics.

Lemma 16. *Let $\vdash \varphi : \mathtt{Prop}$ be an HFA sentence. If $\models \varphi$, then Player 0 wins the operational game $\mathcal{G}(\varphi)$.*

Proof. Let φ_0 be an HFA sentence. We describe a winning strategy of $\mathcal{G}(\varphi_0)$. During the play, Player 0 annotates each labelled μ-binders $\mu^\ell x.\psi$ of type τ with an ordinal γ; we write $\mu^\ell_\gamma x.\psi$ for the annotation. The semantics of $\mu_\gamma x.\psi$ is $[\![\lambda x.\psi]\!]^\gamma(\bot)$, the γ-th stage of the iteration calculating the least fixed-point.

Player 0 ensures during the play that $[\![\varphi]\!] = \top$ where $\langle\varphi\rangle^\ell_T$ is the current node. The initial formula φ_0 satisfies the condition by the assumption. If the current formula is $\varphi_1 \vee \varphi_2$, then $[\![\varphi_1 \vee \varphi_2]\!] = \top$; Proponent chooses the branch i such that $[\![\varphi_i]\!] = \top$. Assume that the current formula is $(\mu^\ell_\gamma x.\varphi)\,\vec{\psi}$. Then the formula in the next step is $\varphi\{(\mu^{\ell'}_{\gamma'} x.\varphi)/x\}\,\vec{\psi}$ for some γ'. We define γ' as the minimum ordinal such that

$$[\![\lambda x.\varphi_0]\!]^{\gamma'+1}(\bot)(\overrightarrow{[\![\psi]\!]}) = \top;$$

such an ordinal exists since $[\![(\mu^\ell_\gamma X.\varphi)\,\vec{\psi}]\!] = \top$.

Claim. $\gamma' < \gamma$.

Proof. If γ is a successor ordinal, i.e. $\gamma = \gamma_0 + 1$, obviously $\gamma' \le \gamma_0 < \gamma$. Assume that γ is a limit ordinal. Then

$$\bigvee_{\gamma_0 < \gamma} [\![\lambda x.\varphi]\!]^{\gamma_0}(\bot)(\overrightarrow{[\![\psi]\!]}) \;=\; [\![(\mu^\ell_\gamma.\varphi)\,\vec{\psi}]\!] \;=\; \top.$$

Hence $[\![\lambda x.\varphi]\!]^{\gamma_0}(\bot)(\overrightarrow{[\![\psi]\!]}) = \top$ for some $\gamma_0 < \gamma$, which implies $[\![\lambda x.\varphi]\!]^{\gamma_0+1}(\bot)(\overrightarrow{[\![\psi]\!]}) = \top$. □

Unlabelled μ-formulas $(\mu x.\varphi)\,\vec{\psi}$ can be treated similarly.

Assume an infinite play following the above strategy. By the definition of the strategy, each μ-label in the call tree T is associated with an ordinal. By construction, $(\ell, \mu, \ell') \in T$ ($\ell \neq 0$) implies $\gamma > \gamma'$, where γ and γ' are ordinals associated to ℓ and ℓ', respectively. Hence the call tree has no infinite μ-path. This means that the strategy is winning. □

Theorem 17. *Player 0 wins $\mathcal{G}(\varphi)$ if and only if $\models \varphi$.*

Proof. If $\models \varphi$, then Player 0 wins the game by Lemma 16. Assume otherwise. Then $\models \neg\varphi$ and thus Player 0 wins $\mathcal{G}(\neg\varphi)$ by Lemma 16. Since $\mathcal{G}(\neg\varphi)$ is essentially the dual game $\mathcal{G}(\varphi)^{\perp}$, Player 1 wins $\mathcal{G}(\varphi)$. □

Corollary 18. *The game $\mathcal{G}(\varphi)$ is determined.*

We are ready to prove the main result.

Theorem 19. *HFA is Δ^1_2.*

Proof. Each node $\langle\psi\rangle^{\ell}_{T}$ of the operational game has only finite information and thus can be coded by natural numbers. The coding system can be chosen so that the game graph is computable. Then a strategy is represented by a function $\omega \to \omega$ on natural numbers. Given a play, the unique infinite path in the call tree is an infinite sequence of natural numbers, which can be naturally represented by a function $\omega \to \omega$. The predicate $\mathit{IsPath}(f, g, s, \llcorner\varphi\lrcorner)$ that checks if $s : \omega \to \omega$ is the infinite path of the call tree of the play generated by the strategies f and g and started from $\langle\varphi\rangle^0_{\emptyset}$ is arithmetic: $\forall n.\exists m.$"$(s(n), \_, s(n+1)) \in T_m(f, g, \llcorner\varphi\lrcorner)$", where $T_m(f, g, \llcorner\varphi\lrcorner)$ is the T-component of the m-th node in the play determined by f, g and $\langle\varphi\rangle^0_{\emptyset}$, which is obviously computable. Hence Player 0 wins the game $\mathcal{G}(\varphi)$ if and only if

$$\exists f.\forall g.\forall s.\ \text{"}g\text{ first violates the rule when starting from }\langle\varphi\rangle^0_{\emptyset}\text{"}$$
$$\vee \mathit{IsPath}(f, g, s, \llcorner\varphi\lrcorner) \Rightarrow \exists m.\text{"}(s(0), v, s(1)) \in T_m(f, g, \llcorner\varphi\lrcorner)\text{"}$$

This shows that *HFA* is Σ^1_2.

Since the game is determined, we can exchange the quantifiers $\exists f$ and $\forall g$ without changing the meaning. Furthermore, since the infinite path of a call tree is unique, one can existentially quantify the path s. Hence

$$\forall g.\exists f.\exists s.\ \text{"}g\text{ first violates the rule when starting from }\langle\varphi\rangle^0_{\emptyset}\text{"}$$
$$\vee\Big(\mathit{IsPath}(f, g, s, \llcorner\varphi\lrcorner) \wedge \exists m.\text{"}(s(0), v, s(1)) \in T_m(f, g, \llcorner\varphi\lrcorner)\text{"}\Big)$$

is another characterisation of the winning region of $\mathcal{G}$. Hence *HFA* is Π^1_2. □

4.3 Strictness of the Hierarchy

This subsection proves the strictness of the hierarchy $\{HFA_n\}_n$. The key observation is that there exists an order-$(n+1)$ HFA formula that defines the set of (codes of) true order-n HFA sentences. Then the strictness of the hierarchy follows from Tarski's undefinability theorem.

Let us fix a coding function $\llcorner - \lrcorner$, which maps each syntactic objects (such as formulas, types, type environments and type judgements) to natural numbers. We assume that $\llcorner - \lrcorner$ is injective and that each syntactic construction is computable (e.g. there exists a computable function $app : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $app(\llcorner\varphi\lrcorner, \llcorner\psi\lrcorner) = \llcorner\varphi\,\psi\lrcorner$). We shall construct an order-$(n+1)$ formula that defines the set

$$HFA_n \quad := \quad \{\, \llcorner\varphi\lrcorner \mid\ \models \varphi,\ order(\varphi) \leq n \,\}.$$

Let us first recall the semantics of formulas. Given a formula $\Gamma \vdash \varphi : \tau$, we have defined a (monotone) mapping $[\![\Gamma]\!] \ni \varrho \mapsto [\![\varphi]\!]_{\varrho} \in [\![\tau]\!]$; hence the semantic interpretation induces a family of functions $I_{\Gamma,\tau} : \mathbb{N} \to [\![\Gamma]\!] \to [\![\tau]\!]$,

$$I_{\Gamma,\tau}(i, \varrho) := \begin{cases} [\![\varphi]\!]_{\varrho} & (\text{if } i = \llcorner\varphi\lrcorner \text{ for some } \Gamma \vdash \varphi : \tau) \\ \top_{[\![\tau]\!]} & (\text{otherwise}), \end{cases}$$

parameterised by Γ and τ. The reference to the semantics $[\![-]\!]_{-}$ in the above definition can be removed, by unfolding the definition of $[\![-]\!]_{-}$ and invoking $I_{\Gamma',\tau'}$ of appropriate types if necessarily. Hence the family $(I_{\Gamma,\tau})_{\Gamma,\tau}$ can be defined by mutual induction.

This inductive definition is almost satisfactory: the inductive definition could be directly describable in HFA, if the family $(I_{\Gamma,\tau})_{\Gamma,\tau}$ were a finite family. The set of order-n types is, however, countably infinite, as well as the set of order-n type environments. This is the problem.

To overcome the problem, we introduce a "generic" type ϑ and a "generic" type environment Θ in which every order-n formula $\Gamma \vdash \varphi : \tau$ can be interpreted. That means, we give an alternative interpretation for order-n formulas

$$(\!|\Gamma \vdash \varphi : \tau|\!)^n_{\epsilon} \in [\![\vartheta]\!], \qquad \text{for each } \epsilon \in [\![\Theta]\!],$$

which induces a function $I^n : \mathbb{N} \to [\![\Theta]\!] \to [\![\theta]\!]$ such that

$$I^n(\llcorner\Gamma \vdash \varphi : \tau\lrcorner)(\epsilon) = (\!|\Gamma \vdash \varphi : \tau|\!)^n_{\epsilon}.$$

Since the type of $(\!|\Gamma \vdash \varphi : \tau|\!)^n$ is independent of Γ and τ, the new interpretation does not suffer from the above problem of infinity. In fact, it is not difficult to see that I^n is definable by an order-$(n+1)$ formula, once the interpretation $(\!|-|\!)^n$ is given.

We formalise the above idea.

Remark 20. We need some complicated definitions of elements in the semantics domain. For simplicity of the presentation, we use λ-calculus notations such as $\lambda y.(\bigwedge_{i \in I} x_i)\, y$, which are justified by the fact that the category of posets and monotone functions is a CCC, i.e. it supports all λ-calculus constructs. Although we use the same symbols for syntactic constructs of HFA and meta-theoretic λ-calculus notations, the meaning should be clear from the context. □

The "generic" order-n type $[n]$ is defined as follows:[5]

$$[1] := \mathtt{Nat} \to \mathtt{Prop}$$
$$[n] := \mathtt{Nat} \times [n-1] \to \mathtt{Prop}, \qquad n \geq 2.$$

There is an isomorphism $\langle -, - \rangle_{[n]} : [\![[n]]\!] \times [\![[n]]\!] \to [\![[n]]\!]$ defined for $n \geq 2$ by

$$\langle x, y \rangle_{[n]}\,(k, z) := \begin{cases} x\,(k', z) & (\text{if } k = 2k') \\ y\,(k', z) & (\text{if } k = 2k' + 1), \end{cases}$$

[5] We slightly extend the logic by products/pairs in argument positions, which can be removed by Currying $(A \times B \to C) \cong (A \to B \to C)$.

which is induced from

$$\begin{aligned}&(\mathbb{N} \times [\![[n-1]]\!] \to \Omega) \times (\mathbb{N} \times [\![[n-1]]\!] \to \Omega)\\ \cong\ & \big((\mathbb{N} \times [\![[n-1]]\!]) + (\mathbb{N} \times [\![[n-1]]\!])\big) \to \Omega\\ \cong\ & (\mathbb{N}+\mathbb{N}) \times [\![[n-1]]\!] \to \Omega\\ \cong\ & \mathbb{N} \to [\![[n-1]]\!] \to \Omega;\end{aligned}$$

the definition of $\langle -,- \rangle_{[1]}$ is similar. Then a finite list of elements in $[\![[n]]\!]$ can also be embedded into $[\![[n]]\!]$:

$$\langle x_1; x_2; \dots; x_k \rangle_{[n]} := \langle x_1, \langle x_2, \dots \langle x_k, \mathsf{Nil}_{[n]} \rangle_{[n]} \dots \rangle_{[n]} \rangle_{[n]},$$

where $\mathsf{Nil}_{[n]} = \bot$.

The "generic" type environment for order-n formulas is $\Theta^n = (e_1 : \mathsf{Nat}, e_2 : [n])$. Hence $[\![\Theta^n]\!] \cong [\![\mathsf{Nat}]\!] \times [\![[n]]\!]$. One can code a finite list of elements in $[\![\mathsf{Nat}]\!] + [\![[n]]\!]$. For $m \in \mathbb{N}$, $v \in [\![[n]]\!]$ and $(\epsilon_1, \epsilon_2) \in \mathbb{N} \times [\![[n]]\!]$, let

$$\begin{aligned} m \overset{\mathsf{Nat}}{::} (\epsilon_1, \epsilon_2) &\quad:=\quad (\langle m, \epsilon_1 \rangle_{\mathsf{Nat}}, \epsilon_2)\\ v \overset{[n]}{::} (\epsilon_1, \epsilon_2) &\quad:=\quad (\epsilon_1, \langle v, \epsilon_2 \rangle_{[n]}),\end{aligned}$$

where $\langle -,- \rangle_{\mathsf{Nat}} : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ is a computable bijection. Then a finite list $(v_i)_{1 \le i \le k} \in \prod_{1 \le i \le k} [\![\vartheta_i]\!]$, where $\vartheta_i = \mathsf{Nat}$ or $[n]$, defines an element of $\mathbb{N} \times [\![[n]]\!]$ by

$$\langle v_1; \dots; v_k \rangle_{\mathsf{Nat}\times[n]} := v_1 \overset{\vartheta_1}{::} (v_2 \overset{\vartheta_2}{::} (\dots (v_k \overset{\vartheta_k}{::} \mathsf{Nil}_{\mathsf{Nat}\times[n]}) \dots)),$$

where $\mathsf{Nil}_{\mathsf{Nat}\times[n]} := (0, \bot)$.

We define a logical relation $(\sim^n_\tau) \subseteq [\![\tau]\!] \times [\![[n]]\!]$ parameterised by τ and n such that $\mathit{order}(\tau) \le n$:[6]

$$\begin{aligned} &x \sim^n_{\mathsf{Prop}} y \overset{\text{def}}{\Leftrightarrow} x = y\,(0, \bot)\\ &x \sim^n_{\vartheta\to\tau} y \overset{\text{def}}{\Leftrightarrow} x\,v \sim^n_\tau (\lambda e^{\mathbb{N}\times[n-1]}. y\,(w \overset{\vartheta}{::} e)) \text{ for every } v \approx^{n-1}_\vartheta w\end{aligned}$$

where $(\overset{\vartheta}{::}) = (\overset{[n-1]}{::})$ for every $\vartheta \ne \mathsf{Nat}$[7] and the relation $(\approx^n_\vartheta) \subseteq [\![\vartheta]\!] \times (\mathbb{N} + [\![[n]]\!])$ is defined by

$$\begin{aligned} m \approx^n_{\mathsf{Nat}} w &\overset{\text{def}}{\Leftrightarrow} m = w \in \mathbb{N}\\ v \approx^n_\tau w &\overset{\text{def}}{\Leftrightarrow} v \sim^n_\tau w \in [\![[n]]\!].\end{aligned}$$

In particular, for $\tau = \vartheta_1 \to \cdots \to \vartheta_k \to \mathsf{Prop}$, we have $x \sim^n_\tau y$ if and only if $v_i \approx^{n-1}_{\vartheta_i} w_i$ $(i = 1, \dots, k)$ implies

$$x\,v_1\, \dots\, v_k = y\,\langle w_1; \dots; w_k \rangle_{\mathsf{Nat}\times[n]}.$$

This relation is closed under limits.

Lemma 21. *Assume a set I and $x_i \sim^n_\tau y_i$ for every $i \in I$. Then*

$$\Big(\bigwedge_{i\in I} x_i\Big) \sim^n_\tau \Big(\bigwedge_{i\in I} y_i\Big) \quad \textit{and} \quad \Big(\bigvee_{i\in I} x_i\Big) \sim^n_\tau \Big(\bigvee_{i\in I} y_i\Big).$$

[6] Here we assume that $n \ge 2$. The definition for $n = 1$ sightly differs since ϑ must be Nat in this case.

[7] Strictly speaking, $\overset{\vartheta}{::}$ is ambiguous if ϑ coincides with $[m]$ for some $m < n$, but the appropriate meaning should be clear from the context.

Proof. By induction on τ. We prove the former. Note that the limit of functions is the point-wise limit, i.e. $(\bigwedge_{i\in I} x_i)\, v = \bigwedge_{i\in I}(x_i\, v)$. (Since the proof only relies on this fact, the latter case can be proved similarly.)

- Case Prop: By the assumption, $x_i = y_i\, 0\, \bot$ for each $i \in I$. Hence $(\bigwedge_{i\in I} y_i)\, 0\, \bot = \bigwedge_{i\in I}(y_i\, 0\, \bot) = \bigwedge_{i\in I} x_i$.
- Case $\sigma \to \tau$ $(\sigma \ne \mathsf{Nat})$: By the assumption, we have $(x_i\, v) \sim^n_\tau (\lambda mz. y_i\, m\, \langle w, z \rangle_{[n-1]})$ for each $i \in I$ and v, w such that $v \sim^{n-1}_\sigma w$. For each m and z,

$$\begin{aligned} &\Big(\bigwedge_{i\in I}(\lambda mz. y_i\, m\, \langle w, z \rangle_{[n-1]})\Big)\, m\, z\\ &= \bigwedge_{i\in I} \Big((\lambda mz. y_i\, m\, \langle w, z \rangle_{[n-1]})\, m\, z\Big)\\ &= \bigwedge_{i\in I} (y_i\, m\, \langle w, z \rangle_{[n-1]})\\ &= \Big(\bigwedge_{i\in I} y_i\Big)\, m\, \langle w, z \rangle_{[n-1]}\\ &= \Big(\lambda mz. \Big(\bigwedge_{i\in I} y_i\Big)\, m\, \langle w, z \rangle_{[n-1]}\Big)\, m\, z.\end{aligned}$$

By extensionality,

$$\bigwedge_{i\in I}(\lambda mz. y_i\, m\, \langle w, z \rangle_{[n-1]}) \quad = \quad \lambda mz. (\bigwedge_{i\in I} y_i)\, m\, \langle w, z \rangle_{[n-1]}.$$

Hence, by the induction hypothesis,

$$\begin{aligned}(\bigwedge_{i\in I} x_i)\, v = \bigwedge_{i\in I}(x_i\, v) \sim^n_\tau &\bigwedge_{i\in I}(\lambda mz. y_i\, m\, \langle w, z \rangle_{[n-1]})\\ &= \lambda mz. (\bigwedge_{i\in I} y_i)\, m\, \langle w, z \rangle_{[n-1]}.\end{aligned}$$

Since (v, w) is an arbitrary pair such that $v \sim^{n-1}_\sigma w$, we have $\bigwedge_{i\in I} x_i \sim^n_{\sigma\to\tau} (\bigwedge_{i\in I} y_i)$.
- Case $\mathsf{Nat} \to \tau$: Similar to the above case.

□

The relation $\approx^n_\vartheta$ can naturally be extended to sequences, i.e. $(\sim^n_\Gamma) \subseteq [\![\Gamma]\!] \times ([\![\mathsf{Nat}]\!] \times [\![[n]]\!])$ parameterised by order-n type environments Γ: for $\Gamma = (x_1 : \vartheta_1, \dots, x_k : \vartheta_k)$,

$$\varrho \approx^n_\Gamma \epsilon \overset{\text{def}}{\Leftrightarrow} \epsilon = \langle w_k; \dots; w_1 \rangle_{\mathsf{Nat}\times[n]} \text{ and } \varrho(x_i) \approx^n_\vartheta w_i \text{ for all } i.$$

We can now formally state the requirement for the alternative interpretation $(\!|\Gamma \vdash \varphi : \tau|\!)^n : \mathbb{N} \times [\![[n]]\!] \to [\![[n]]\!]$:

$$[\![\Gamma \vdash \varphi : \tau]\!]_\varrho \sim^n_\tau (\!|\Gamma \vdash \varphi : \tau|\!)^n_\epsilon \text{ for every } \varrho \approx^n_\Gamma \epsilon.$$

The definition of $(\!|-|\!)^n$ is rather straightforward. For notational convenience, we abbreviate $(\!|\Gamma \vdash \varphi : \tau|\!)^n$ as $(\!|\varphi|\!)^n$. The definition uses the order-shifting functions

$$\Downarrow_n \;:\; [\![[n+1]]\!] \to [\![[n]]\!] \qquad \Uparrow_n \;:\; [\![[n]]\!] \to [\![[n+1]]\!]$$

that satisfies

- $\Downarrow_n (\Uparrow_n x) = x$,
- $v \sim^n_\tau w$ implies $v \sim^{n+1}_\tau (\Uparrow_n w)$, and
- $v \sim^{n+1}_\tau w$ and $\mathit{order}(\tau) \le n$ implies $v \sim^n_\tau (\Downarrow_n w)$.

Such operations can be defined for $n \geq 2$ by

$$\Uparrow_n(v) := \lambda(a_1, a_2)^{\mathbb{N}\times[n]}.v\,(a_1, (\Downarrow_{n-1}\ a_2))$$
$$\Downarrow_n(v) := \lambda(a_1, a_2)^{\mathbb{N}\times[n-1]}.v\,(a_1, (\Uparrow_{n-1}\ a_2))$$

and similarly for $n = 1$. The definition includes the rules

$$\llparenthesis \varphi_1\, \varphi_2^{\vartheta} \rrparenthesis^n_\epsilon(a) := \llparenthesis \varphi_1 \rrparenthesis^n_\epsilon((\Downarrow_{n-1}\llparenthesis \varphi_2 \rrparenthesis^n_\epsilon) \overset{\vartheta}{::} a)$$
$$\llparenthesis \varphi_1 \wedge \varphi_2 \rrparenthesis^n_\epsilon(a) := \llparenthesis \varphi_1 \rrparenthesis^n_\epsilon(a) \wedge \llparenthesis \varphi_2 \rrparenthesis^n_\epsilon(a)$$
$$\llparenthesis \lambda x^{\vartheta}.\varphi \rrparenthesis^n_\epsilon(a_0 \overset{\vartheta}{::} a) := \llparenthesis \varphi \rrparenthesis^n_{(\Uparrow_{n-1}\ a_0)\overset{\vartheta}{::}\epsilon}(a)$$
$$\llparenthesis \mu X^{\tau}.\varphi \rrparenthesis^n_\epsilon(a) := \left(\mathsf{lfp}\,\lambda Y^{\llbracket[n]\rrbracket}.\llparenthesis \varphi \rrparenthesis^n_{Y\overset{\tau}{::}\epsilon}\right)(a).$$

Lemma 22. *For every order-n formula $\Gamma \vdash \varphi : \tau$, we have $\llbracket \varphi \rrbracket_\varrho \sim^n_\tau \llparenthesis \varphi \rrparenthesis^n_\epsilon$ for every $\varrho \approx^n_\Gamma \epsilon$.*

Proof. By induction on φ. Assume $\varrho \approx^n_\Gamma \epsilon$.

Assume $\varphi = \varphi_1\,\varphi_2$. Then $\Gamma \vdash \varphi_1 : \vartheta \to \tau$ and $\Gamma \vdash \varphi_2 : \vartheta$ for some ϑ. Consider, for example, the case that $\vartheta \neq \mathtt{Nat}$. Then, by the induction hypothesis, $\llbracket \varphi_2 \rrbracket_\varrho \sim^n_\vartheta \llparenthesis \varphi_2 \rrparenthesis^n_\epsilon$. Since $\mathit{order}(\vartheta) < n$, we have $\llbracket \varphi_2 \rrbracket_\varrho \sim^{n-1}_\vartheta \Downarrow_{n-1}\llparenthesis \varphi_2 \rrparenthesis^n_\epsilon$. As $\llbracket \varphi_1 \rrbracket_\varrho \sim^n_{\vartheta\to\tau} \llparenthesis \varphi_1 \rrparenthesis^n_\epsilon$ by the induction hypothesis, we have $\llbracket \varphi_1 \rrbracket_\varrho(\llbracket \varphi_2 \rrbracket_\varrho) \sim^n_\tau \lambda a.\llparenthesis \varphi_1 \rrparenthesis^n_\epsilon((\Downarrow_{n-1} \llparenthesis \varphi_2 \rrparenthesis^n_\epsilon) \overset{\vartheta}{::} a)$. Thin means $\llbracket \varphi_1\,\varphi_2 \rrbracket_\varrho \sim^n_\tau \llparenthesis \varphi_1\,\varphi_2 \rrparenthesis^n_\epsilon$.

Assume $\varphi = \lambda x^{\vartheta}.\psi$. Then $\tau = \vartheta \to \sigma$ and $\Gamma, x : \vartheta \vdash \psi : \sigma$ for some σ. By definition, it suffices to show that $\llbracket \lambda x^{\vartheta}.\psi \rrbracket_\varrho(v) \sim^n_\sigma (\lambda z.\llparenthesis \lambda x^{\vartheta}.\psi \rrparenthesis_\epsilon\,(w \overset{\vartheta}{::} z))$ for every $v \approx^{n-1}_\vartheta w$. By calculating both sides, this is equivalent to $\llbracket \psi \rrbracket_{\varrho[v/x]} \sim^n_\sigma \lambda z.\llparenthesis \psi \rrparenthesis^n_{(\Uparrow_{n-1}\ w)\overset{\vartheta}{::}\epsilon}\ z = \llparenthesis \psi \rrparenthesis^n_{(\Uparrow_{n-1}\ w)\overset{\vartheta}{::}e}$, which follows from the induction hypothesis.

Assume $\varphi = \mu X^{\tau}.\psi$. Then $\Gamma, X : \tau \vdash \psi : \tau$. Let $f = \llbracket \lambda X.\psi : \tau \to \tau \rrbracket_\varrho : \llbracket \tau \rrbracket \to \llbracket \tau \rrbracket$ and $g = \lambda Y^{\llbracket[n]\rrbracket}.\llparenthesis \psi \rrparenthesis^n_{Y\overset{\tau}{::}e} : \llbracket[n]\rrbracket \to \llbracket[n]\rrbracket$. Then $\llbracket \varphi \rrbracket_\varrho = \mathsf{lfp}\, f$ and $\llparenthesis \varphi \rrparenthesis^n_\epsilon = \mathsf{lfp}\, g$. It suffices to show that $f^{\gamma}(\bot) \sim^n_\tau g^{\gamma}(\bot)$ for every ordinal number γ. The base case is $\bot \sim^n_\tau \bot$, which is easy. The case of limit ordinals follows from Lemma 21. We prove that $v \sim^n_\tau w$ implies $f(v) \sim^n_\tau g(w)$. If $v \sim^n_\tau w$, then $\varrho[v/X] \approx^n_{\Gamma,X:\tau} (w \overset{\tau}{::} \epsilon)$. Hence, by the induction hypothesis, $f(v) = \llbracket \psi \rrbracket_{\varrho[v/X]} \sim^n_{\Gamma,X:\tau} \llparenthesis \psi \rrparenthesis^n_{w\overset{\tau}{::}e} = g(w)$ as required.

Other cases are similar. □

Corollary 23. $\llbracket \varphi \rrbracket_\emptyset = \llparenthesis \varphi \rrparenthesis^n_{\mathsf{Nil}_{\mathtt{Nat}\times[n]}}(\mathsf{Nil}_{\mathtt{Nat}\times[n-1]})$ *for every order-n sentence* $\emptyset \vdash \varphi : \mathtt{Prop}$.

Therefore it suffices to show that $\llparenthesis - \rrparenthesis^n$ is definable by an order-$(n+1)$ HFA formula $I_n : \mathtt{Nat} \to \mathtt{Nat} \to [n] \to [n]$. This formula is obtained from the definition of $\llparenthesis - \rrparenthesis^n$ by replacing $\llparenthesis \varphi \rrparenthesis^n_\epsilon$ with $I_n \llcorner \varphi \lrcorner\, \epsilon$, because operations appearing in the definition such as $\Uparrow_{n-1}$ and $\Downarrow_{n-1}$ are definable by order-$(n+1)$ formulas.

Theorem 24. *HFA_n is definable by an order-$(n+1)$ formula.*

No order-n formula defines HFA_n by Tarski's undefinability theorem.[8] Hence the set of order-$(n+1)$ formulas is strictly more expressive than that of order-n formulas. The main theorem of this subsection is a direct consequence of this argument.

Theorem 25. *$HFA_n <_m HFA_{n+1}$.*

Proof. Trivially $HFA_n \leq_m HFA_{n+1}$. We prove that HFA_{n+1} is not many-one reducible to HFA_n. Assume for contradiction that HFA_{n+1} is reducible to HFA_n. Then there exists a Σ^0_1-formula $\varphi(x, y)$ such that, for every order-$(n+1)$ sentence ξ, we have $\models \xi$ if and only if there exists an order-n sentence ζ such that $\models \zeta$ and $\models \varphi(\llcorner \xi \lrcorner, \llcorner \zeta \lrcorner)$. Letting T^n be the formula in Theorem 24, the order-$(n+1)$ formula $\exists y.T^n(y) \wedge \varphi(x, y)$ defines HFA_{n+1}, which contradicts to Tarski's undefinability theorem. □

Remark 26. A slight modification of the above argument shows that $HFA_n <_T HFA_{n+1}$ (i.e. HFA_{n+1} is not Turing reducible to HFA_n) and even that HFA_{n+1} is not arithmetical in HFA_n. □

5 Extensions of Constrained Horn Clauses

This section studies some extensions of the satisfiability problem for *constrained Horn clauses* (*CHCs* for short) that have been studied in the context of program verification [1–3].

5.1 Constrained Horn Clauses

The theory of constrained Horn clauses is usually parameterised by the background theory, but we fix the background theory to the quantifier-free linear arithmetic. It is a fairly weak and commonly used theory. We would like to show that generalised CHC are hard even for a weak background theory.

We shall consider following forms of "clauses":

$$\begin{array}{lccl}
\text{(Base)} & \varphi \wedge H_1(\vec{x}_1) \wedge \cdots \wedge H_n(\vec{x}_n) & \to & G(\vec{y}) \\
(\vee) & \varphi \wedge H_1(\vec{x}_1) \wedge \cdots \wedge H_n(\vec{x}_n) & \to & G_1(\vec{y}_1) \vee G_2(\vec{y}_2) \\
(\exists) & \varphi \wedge H_1(\vec{x}_1) \wedge \cdots \wedge H_n(\vec{x}_n) & \to & \exists z.G(z, \vec{y}) \\
(\mathit{wf}) & & \mathit{wf}(H) &
\end{array}$$

Here $H_1, \dots, H_n$ are predicate variables, G is a predicate variable or $\bot$ and H is a predicate variable of arity 2. (Base) is the standard constrained Horn clause, and $(\vee)$ and $(\exists)$ are extensions allowing $\vee$ and $\exists$ at the head (i.e. the right-hand-side of $\to$); $\mathit{wf}(H)$ requires that the binary predicate H is well-founded, i.e. there is no infinite sequence $a_0 a_1 a_2 \dots$ such that all adjacent pairs of elements are related by H.

The meaning of each clause should be clear. The *satisfiability problem* asks, given a finite set of clauses, whether

[8] One has to check that HFA_n satisfies the requirements of the theorem. The diagonal argument applies to HFA_n since it contains first-order arithmetic. For closure under negation, see Remark 11.

there is a valuation to predicate variables that satisfies all the clauses.

Let us write $CHC[\exists, wf]$ for the satisfiability problem for finite sets of ($\exists$)- and (wf)-clauses in addition to (Base)-clauses. The meaning of $CHC[\vee]$ and others should be clear. In this notation, the problem used in Beyene *et al.* [1] is $CHC[\exists, wf]$.

5.2 Σ^1_2-completeness of $CHC[\exists, wf]$

We show both $CHC[\exists, wf]$ and $CHC[\vee, wf]$ are Σ^1_2-complete. Hence they are strictly harder than *Verif*. Since a ($\vee$)-clause can be expressed by using a ($\exists$)-clause, $CHC[\vee, wf] \le_m CHC[\exists, wf]$. We show that

1. $CHC[\exists, wf]$ is Σ^1_2, and
2. $CHC[\vee, wf]$ is Σ^1_2-hard.

As a consequence $CHC[\vee, wf] \equiv_m CHC[\exists, wf]$.

The former claim is obvious. A finite set of clauses in $CHC[\exists, wf]$, say $C = \{\Phi_1, \dots, \Phi_n\} \cup \{wf(H_1), \dots, wf(H_m)\}$ with free predicate variables $\vec{P}$, is satisfiable if and only if the Σ^1_2-formula

$$\exists \vec{P}. \left(\bigwedge_{1 \le i \le n} \forall \vec{x}_i . \Phi_i \right) \\ \wedge \left(\bigwedge_{1 \le j \le m} \forall f^{\mathsf{Nat} \to \mathsf{Nat}} . \exists x^{\mathsf{Nat}} . \neg H_j(f(x), f(x+1)) \right)$$

is true. Since the set of codes of true Σ^1_2-formulas is a Σ^1_2-set and the above translation is computable, we conclude $CHC[\exists, wf] \in \Sigma^1_2$.

We prove the harder part.

Lemma 27. *$CHC[\vee, wf]$ is Σ^1_2-hard.*

Proof. Consider the following decision problem:

$$\left\{ \llcorner \varphi \lrcorner \;\middle|\; \begin{array}{l} \varphi(x, y, Z) : \Sigma^0_1\text{-formula} \\ [\![\lambda xy.\varphi]\!]_{[A/Z]} \text{ is well-founded for some } A \subseteq \mathbb{N} \end{array} \right\}.$$

Here x and y are natural number variables, Z is a unary predicate variable, and φ has no other free variable. This problem is Σ^1_2-complete [11, Theorem XXXVII, §16.4, p.416]. We reduce this problem to $CHC[\vee, wf]$.

Logical constructs in Σ^0_1-formulas can be "simulated" by CHCs. We construct a finite set of CHCs C_ψ with distinguished predicate symbols Z and P_ψ that satisfies the following conditions:

- For every $A \subseteq \mathbb{N}$, there exists a solution ϱ of C_ψ such that $\varrho(Z) = A$ and $\varrho(P_\psi) = [\![\psi]\!]_{[A/Z]}$.
- Every solution ϱ of C_ψ satisfies $[\![\psi]\!]_{[\varrho(Z)/Z]} \subseteq \varrho(P_\psi)$.

We define C_ψ by induction on ψ. We can assume without loss of generality that ψ is in negation normal form. The most important case is $\psi(x) = \neg Z(x)$, where $C_{\neg Z(x)}$ is

$$\{ \mathtt{true} \to P_{\neg Z(x)}(x) \vee Z(x),\; P_{\neg Z(x)}(x) \wedge Z(x) \to \mathtt{false} \}.$$

The other cases are rather straightforward. For example, $C_{\psi_1 \vee \psi_2}$ consists of the rules in C_{ψ_1} and in C_{ψ_2} with the following additional rules:

$$P_{\psi_1}(\vec{x}) \to P_{\psi_1 \vee \psi_2}(\vec{x}) \qquad P_{\psi_2}(\vec{x}) \to P_{\psi_1 \vee \psi_2}(\vec{x}).$$

For another example, the representation of a bounded quantifier $\forall y \le t.\psi$ is given by

$$\begin{aligned} P_\psi(\vec{x}, 0) &\to H(\vec{x}, 0) \\ y = \mathsf{S}(y') \wedge P_\psi(\vec{x}, y) \wedge H(\vec{x}, y') &\to H(\vec{x}, y) \\ H(\vec{x}, t) &\to P_{\forall y \le t.\psi}(\vec{x}) \end{aligned}$$

in addition to C_ψ. Finally $\exists x.\psi$ can be coded by

$$P_\psi(x, \vec{y}) \to P_{\exists x.\psi}(\vec{y}).$$

It is easy to see that this construction satisfies the requirements.

Then

$$\begin{aligned} & C_\varphi \cup \{wf(P_\varphi)\} \text{ is satisfiable} \\ \Leftrightarrow & \exists A \subseteq \mathbb{N}. \exists B \subseteq \mathbb{N}^2.\ [\![\lambda xy.\varphi]\!]_{[A/Z]} \subseteq B \text{ and } B \text{ is well-founded} \\ \Leftrightarrow & \exists A \subseteq \mathbb{N}.\ [\![\lambda xy.\varphi]\!]_{[A/Z]} \text{ is well-founded.} \end{aligned}$$

For the second equivalence, note that $B \subseteq \mathbb{N}^2$ is well-founded and $B' \subseteq B$ implies B' is well-founded.

Obviously $\llcorner \varphi \lrcorner \mapsto (C_\varphi \cup \{wf(P_\varphi)\})$ is effective. This completes the proof of the lemma. □

Theorem 28. *$CHC[\exists, wf]$ and $CHC[\vee, wf]$ are Σ^1_2-complete.*

A consequence is that $CHC[\exists, wf] \equiv_m CHC[\vee, wf]$, i.e. one can effectively remove ($\exists$)-clauses by using ($\vee$)-clauses preserving the satisfiability. As $CHC[\vee, wf]$ is superficially easier, whether there exists a practical translation could be of practical interest. (Unfortunately the translation given by the proofs are complicated.)

Remark 29. So far we have considered the satisfiability with respect to the standard model $\mathbb{N}$ of natural numbers but the proof is applicable to satisfiability modulo weaker theories as well. The proof only requires that the theory contains a function symbol S and a constant z and the following axioms:

$$(\forall x.\mathsf{z} \ne \mathsf{S}(x)) \wedge (\forall xy.\mathsf{S}(x) = \mathsf{S}(y) \Rightarrow x = y).$$

The constraint language suffices to contain $t = t'$ and $t \ne t'$, where $t, t' ::= \mathsf{z} \mid x \mid \mathsf{S}(t)$. This condition would be satisfied by many background theories that deal with infinite data. □

Remark 30. The *disjunctive well-foundedness predicate dwf* is sometimes used instead of wf (e.g. [1]). A relation H is *disjunctively well-founded* if $H = \bigvee_{i=1}^k H_i$ for some well-founded relations H_i. The satisfaction problem $CHC[\vee, dwf]$ is Σ^1_2-complete since $CHC[\vee, dwf]$ and $CHC[\vee, wf]$ are reducible to each other. Since $wf(H) \Leftrightarrow dwf(H^+)$ (where H^+ is the transitive closure of H) [20], $CHC[\vee, wf]$ can be easily reduced to $CHC[\vee, dwf]$. The other direction is a consequence of the Σ^1_2-hardness of $CHC[\vee, wf]$: for a relation H

on natural numbers, it is not difficult to express $dwf(H)$ by a Σ^1_2-formula. ◻

5.3 Hardness of Subproblems

We have proved that $CHC[\vee, wf]$, the problem studied in [1–3], is strictly more difficult than the verification problem. How about natural subproblems $CHC[\vee]$ and $CHC[wf]$?

Theorem 31. *$CHC[\vee] <_m$ Verif and $CHC[wf] <_m$ Verif.*

Proof. A consequence of Lemmas 32 and 33, which shall be proved below. Recall that *Verif* is Π^1_1-hard and Σ^1_1-hard and thus not in $\Pi^1_1 \cup \Sigma^1_1$ (Proposition 6). ◻

Lemma 32. *$CHC[\vee]$ and $CHC[\exists]$ are Σ^1_1.*

Proof. They are Σ^1_1 because $\{\Phi_1, \dots, \Phi_n\}$ is satisfiable if and only if $\exists\vec{P}.\Phi_1 \wedge \cdots \wedge \Phi_n$. ◻

Lemma 33. *$CHC[wf]$ is Π^1_1-complete.*

Proof. Hardness follows from Proposition 2 since a given Σ^0_1-formula can be simulated by (Base)-constraints (cf. Turing-completeness of the core of prolog).

We show that $CHC[wf]$ is Π^1_1. Let

$$C \;=\; \{\, \Phi_1, \dots, \Phi_n, \Psi_1, \dots, \Psi_m, wf(H_1), \dots, wf(H_k) \,\}$$

be a given set of $CHC[wf]$-clauses. Here Φ_i is of the form $\cdots \longrightarrow H(\vec{x}_i)$ and Ψ_j is of the form $\cdots \longrightarrow \bot$. The constraint Ψ_j as well as $wf(H_\ell)$ is monotone in the sense that, if the valuation ϱ satisfies Ψ_j and ϱ' is less than ϱ, then ϱ' also satisfies Ψ_j. So the problem is whether the minimum solution to $\{\Phi_1, \dots, \Phi_n\}$ satisfies other constraints. It is well-known that the minimum solution exists and can be expressed by Σ^0_1-formulas; furthermore the mapping from $\vec{\Phi}$ to formulas is computable. By substituting the obtained Σ^0_1-formulas, what we need to do is the validity checking of Π^0_1-formulas (corresponding to Ψ-constraints) and well-foundedness checking of Σ^0_1-formulas (corresponding to $wf(H_\ell)$). Hence $CHC[wf]$ is Π^1_1. ◻

6 Conclusion

We have studied the hardness of branching-time property verification of Turing-complete programs, as well as logical problems used in verification methods.

We pointed out that the verification problem was, in effect, been studied by Bradfield [4] of which the proofs induce polynomial-time reductions between the verification problem and the validity problem of (first-order) fixed-point arithmetic. This is a remarkable result as all other logical problems studied in this paper are strictly harder than the verification problem. We expect that this result would motivate an extensive study of first-order fixed-point arithmetic in program verification.

The satisfiability problem of constrained Horn clauses with extensions [1–3] is also strictly harder than the verification problem, but it seems a minimal extension to which the verification problem is reducible.

Higher-order fixed-point arithmetic used in [16, 25] are strictly harder than the verification problem, but far easier than the validity problem of second-order arithmetic and the satisfiability problem of extensions of constrained Horn clauses [1–3], at least theoretically. We also showed that the hierarchy $(HFA_n)_n$ is strict by giving an order-$(n+1)$ formula that defines the set of true order-n sentences. As a consequence, formulas of higher-order fixed-point logic cannot be "naturally" transformed into programs nor the priority-typed fragment (i.e. the λY-calculus with priorities [24]). We think this answers the question posed by Walukiewicz [24].

The strictness of the HFA hierarchy and the mismatch to the higher-order verification problem poses a natural question: is there any natural characterisation of the classes? Bradfield [5] established a beautiful connection between the alternation hierarchy of μ-arithmetic and the difference hierarchy over Σ^0_2 using the game quantifier. This approach seems promising. We conjecture that HFA_n is related to the difference hierarchy over Σ^0_{n+1}. Actually higher-order computability, namely *Kolmogorov's R operator*, has been shown to be relevant to the games over Σ^0_3 [10].

Acknowledgments

We would like to thank Naoki Kobayashi and Hiroshi Unno for discussions, and anonymous referees for valuable comments. This work was supported by JSPS Kakenhi 15H05706 and JSPS Kakenhi 20H00577.

References

[1] Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. 2014. A constraint-based approach to solving games on infinite graphs. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 221–234.

[2] Tewodros A. Beyene, Corneliu Popeea, and Andrey Rybalchenko. 2013. Solving Existentially Quantified Horn Clauses. In *The 25th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.), Vol. 8044. Springer, 869–882.

[3] Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. 2015. Horn Clause Solvers for Program Verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday (Lecture Notes in Computer Science)*, Lev D. Beklemishev, Andreas Blass, Nachum Dershowitz, Bernd Finkbeiner, and Wolfram Schulte (Eds.), Vol. 9300. Springer, 24–51.

[4] Julian C. Bradfield. 1998. The Modal μ-Calculus Alternation Hierarchy is Strict. *Theor. Comput. Sci.* 195, 2 (1998), 133–153.

[5] Julian C. Bradfield. 1999. Fixpoint Alternation and the Game Quantifier. In *The 8th EACSL Annual Conference on Computer Science Logic (Lecture Notes in Computer Science)*, Jörg Flum and Mario Rodríguez-Artalejo (Eds.), Vol. 1683. Springer, 350–361.

[6] Florian Bruse. 2014. Alternating Parity Krivine Automata. In *The 39th International Symposium on Mathematical Foundations of Computer*

Science (Lecture Notes in Computer Science), Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik (Eds.), Vol. 8634. Springer, 111–122.

[7] Florian Bruse. 2016. Alternation Is Strict For Higher-Order Modal Fixpoint Logic. In *Proceedings of the 7th International Symposium on Games, Automata, Logics and Formal Verification (EPTCS)*, Domenico Cantone and Giorgio Delzanno (Eds.), Vol. 226. 105–119.

[8] Erich Grädel, Wolfgang Thomas, and Thomas Wilke (Eds.). 2002. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Lecture Notes in Computer Science, Vol. 2500. Springer.

[9] David Harel. 1986. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM* 33, 1 (1986), 224–248.

[10] Thomas John. 1986. Recursion in Kolmogorov's R-Operator and the Ordinal σ_3. *J. Symb. Log.* 51, 1 (1986), 1–11.

[11] Hartley Rogers Jr. 1987. *Theory of recursive functions and effective computability*. MIT Press.

[12] Naoki Kobayashi, Étienne Lozes, and Florian Bruse. 2017. On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In *The 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 246–259.

[13] Naoki Kobayashi, Takeshi Nishikawa, Atsushi Igarashi, and Hiroshi Unno. 2019. Temporal Verification of Programs via First-Order Fixpoint Logic. In *The 26th Static Analysis Symposium (Lecture Notes in Computer Science)*, Bor-Yuh Evan Chang (Ed.), Vol. 11822. Springer, 413–436.

[14] Naoki Kobayashi and C.-H. Luke Ong. 2009. A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes. In *The 24th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 179–188.

[15] Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe. 2017. Higher-Order Program Verification via HFL Model Checking. *CoRR* abs/1710.08614 (2017). arXiv:1710.08614

[16] Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe. 2018. Higher-Order Program Verification via HFL Model Checking. In *The 27th European Symposium on Programming (Lecture Notes in Computer Science)*, Amal Ahmed (Ed.), Vol. 10801. Springer, 711–738.

[17] Dexter Kozen. 2006. *Theory of Computation*. Springer.

[18] Robert S. Lubarsky. 1993. μ-Definable Sets of Integers. *J. Symb. Log.* 58, 1 (1993), 291–313.

[19] Yoji Nanjo, Hiroshi Unno, Eric Koskinen, and Tachio Terauchi. 2018. A Fixpoint Logic and Dependent Effects for Temporal Property Verification. In *The 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, Anuj Dawar and Erich Grädel (Eds.). ACM, 759–768.

[20] Andreas Podelski and Andrey Rybalchenko. 2004. Transition Invariants. In *The 19th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 32–41.

[21] Hiroshi Unno, Yuki Satake, and Tachio Terauchi. 2018. Relatively complete refinement type system for verification of higher-order nondeterministic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 12:1–12:29.

[22] Moshe Y. Vardi. 1991. Verification of Concurrent Programs: The Automata-Theoretic Framework. *Ann. Pure Appl. Log.* 51, 1-2 (1991), 79–98.

[23] Mahesh Viswanathan and Ramesh Viswanathan. 2004. A Higher Order Modal Fixed Point Logic. In *The 15th International Conference on Concurrency Theory (Lecture Notes in Computer Science)*, Philippa Gardner and Nobuko Yoshida (Eds.), Vol. 3170. Springer, 512–528.

[24] Igor Walukiewicz. 2019. Lambda Y-Calculus With Priorities. In *The 34th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 1–13.

[25] Keiichi Watanabe, Takeshi Tsukada, Hiroki Oshikawa, and Naoki Kobayashi. 2019. Reduction from branching-time property verification of higher-order programs to HFL validity checking. In *The 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, Manuel V. Hermenegildo and Atsushi Igarashi (Eds.). ACM, 22–34.