Temporal Verification of Programs via First-Order Fixpoint Logic

Naoki Kobayashi[1], Takeshi Nishikawa[2], Atsushi Igarashi[2], and Hiroshi Unno[3,4]

[1] The University of Tokyo, koba@is.s.u-tokyo.ac.jp
[2] Kyoto University, {igarashi,nishikawa}@fos.kuis.kyoto-u.ac.jp
[3] University of Tsukuba, uhiro@cs.tsukuba.ac.jp
[4] RIKEN AIP

Abstract. This paper presents a novel program verification method based on Mu-Arithmetic, a first-order logic with integer arithmetic and predicate-level least/greatest fixpoints. We first show that linear-time temporal property verification of first-order recursive programs can be reduced to the validity checking of a Mu-Arithmetic formula. We also propose a method for checking the validity of Mu-Arithmetic formulas. The method generalizes a reduction from termination verification to safety property verification and reduces validity of a Mu-Arithmetic formula to satisfiability of CHC, which can then be solved by using off-the-shelf CHC solvers. We have implemented an automated prover for Mu-Arithmetic based on the proposed method. By combining the automated prover with a known reduction and the reduction from first-order recursive programs above, we obtain: (i) for while-programs, an automated verification method for arbitrary properties expressible in the modal μ-calculus, and (ii) for first-order recursive programs, an automated verification method for arbitrary linear-time properties expressible using Büchi automata. We have applied our Mu-Arithmetic prover to formulas obtained from various verification problems and obtained promising experimental results.

1 Introduction

Several researchers have recently advocated the use of fixpoint logics in program verification. The idea at least goes back to the early work of Blass [10], who showed that the weakest preconditions of while-loops can be expressed by using a fixpoint logic. Bjorner et al. [5, 8, 9, 22] advocated a reduction from program verification problems to the satisfiability of Constrained Horn Clauses (CHC), which is essentially the validity checking for a restricted fragment of first-order fixpoint logic. Burn et al. [12] have recently extended the approach to a higher-order extension of Constrained Horn Clauses. Kobayashi et al. [26,40] have shown that temporal verification problems for higher-order functional programs can be reduced to validity checking problems in a higher-order fixpoint logic. Nanjo et al. [32] also proposed an approach to temporal verification based on a fixpoint logic. One of the main advantages common to those approaches is that a fixpoint

logic prover can be used as a common, language-independent backend tool for a variety of verification problems (not only safety properties but also arbitrary regular temporal properties, including liveness). Fixpoint logic provers have not, however, been available yet with the full generality.

Based on the observation above, in the present paper, we propose a method for automatically checking the validity of first-order fixpoint logic formulas. The first-order fixpoint logic (with integer arithmetic) we consider has been studied before albeit in different contexts and with different syntax [11, 30]. Following Bradfield [11], we call the logic Mu-Arithmetic. For while-programs (with unbounded integers), the formulas generated by the aforementioned translations of Kobayashi et al. [26, 40] are actually Mu-Arithmetic formulas. The core logic used by Nanjo et al. [32] is also Mu-Arithmetic. Thus, by combining those previous studies with our procedures for proving Mu-Arithmetic formulas, we can obtain an automated tool for temporal program verification.

Our method, called Mu2CHC, reduces the validity of a Mu-Arithmetic formula to the satisfiability of CHC (in a sound but incomplete manner). The reduction has been inspired by reductions from termination verification to safety property verification [21, 34]. More precisely, we generalize the termination verification method of Fedyukovich et al. [21] to underapproximate a least fixpoint formula by a greatest fixpoint formula. Given a formula φ consisting of both least and greatest fixpoints, we convert it to a stronger formula φ' (in the sense $\varphi' \Rightarrow \varphi$) that consists of only greatest fixpoint formulas. We then transform it to a set $\mathcal{C}$ of CHCs, so that $\mathcal{C}$ is satisfiable if and only if φ' is valid. This provides a sound method for proving the validity of the original Mu-Arithmetic formula. The main advantages of this approach are: (i) the reduction is fairly simple and easy to implement, and (ii) we can use off-the-shelf CHC solvers [13, 27, 37], avoiding replicated work for, e.g., invariant inference.

We have implemented the proposed approach, and confirmed its effectiveness. The benchmark problems used for experiments contain those beyond the capabilities of the existing related tools (such as CHC solvers and program verification tools).

Another main contribution of the present paper is a sound and complete reduction from linear-time properties (expressible using Büchi automata) of first-order recursive programs to the validity of Mu-Arithmetic formulas. This can be considered a generalization of reductions from safety properties of first-order recursive programs to CHC satisfiability (see [6], Section 3.2). Kobayashi et al. [26] have shown a reduction from linear-time properties of higher-order programs to the validity of *higher-order* fixpoint logic formulas, but for first-order recursive programs, their translation yields a formula of a second-order fixpoint logic, not a first-order one. We also show a reduction from the modal μ-calculus model checking of while-programs to the validity of Mu-Arithmetic formulas. Such a reduction can in principle be obtained from the general reduction of Watanabe et al. [40], but our reduction is more direct.

The rest of the paper is organized as follows. Section 2 defines Mu-Arithmetic, the first-order fixpoint logic with integer arithmetic. Section 3 discusses appli-

cations of the fixpoint logic to program verification; in particular, we show that linear-time properties of first-order recursive programs can be reduced to the validity of Mu-Arithmetic formulas. We propose our method Mu2CHC in Section 4. Section 5 reports on the implementation and experimental evaluation of our methods. We discuss related work in Section 6 and conclude the paper in Section 7.

2 First-Order Fixpoint Logic with Integer Arithmetic

This section introduces the first-order fixpoint logic with integer arithmetic. Following Bradfield [11], we call the logic Mu-Arithmetic (though the syntax is different). We define the syntax and semantics of fixpoint logic in the form of *hierarchical equation systems*, following [36].

2.1 Syntax

The set of *formulas*, ranged over by φ is given by:

$$\varphi ::= a_1 \geq a_2 \mid P(a_1, \ldots, a_k) \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \exists x.\varphi \mid \forall x.\varphi$$
$$a ::= x \mid n \mid a_1 \,\mathbf{op}\, a_2$$

Here, P ranges over a set of predicate names. We have only $\geq$ as a primitive predicate; in examples, we shall use other integer predicates like $=$ (which can be expressed in terms of $\geq$ and other logical operators).

A *hierarchical equation system* (HES) Φ is a pair $(\mathcal{E}, \mathcal{H})$. Here, $\mathcal{E}$ is a set of equations of the form $\{P_1(\widetilde{x}_1) = \varphi_1, \cdots, P_m(\widetilde{x}_m) = \varphi_n\}$, where $\widetilde{x}$ stands for a sequence of variables, and $\mathcal{H}$ is a sequence $(\mathcal{P}_k, \alpha_k); \cdots; (\mathcal{P}_1, \alpha_1)$, where $\mathcal{P}_k, \ldots, \mathcal{P}_1$ are mutually disjoint sets of predicate names such that $\mathcal{P}_k \cup \cdots \cup \mathcal{P}_1 = \{P_1, \ldots, P_m\}$, and $\alpha_i \in \{\mu, \nu\}$. We write $\mathcal{P}_\Phi$, $\mathcal{P}_{\leq i}$, and $\mathcal{P}_{\geq i}$ for $\bigcup_j \mathcal{P}_j$, $\bigcup_{j \leq i} \mathcal{P}_j$, and $\bigcup_{j \geq i} \mathcal{P}_j$, respectively. We often write $\mathcal{P}$ for $\mathcal{P}_\Phi$ if Φ is clear from the context. We also write $\mathcal{E}_{\mathcal{P}_i}$, or just $\mathcal{E}_i$ to denote the subset of equations $\{P(\widetilde{x}) = \varphi \in \mathcal{E} \mid P \in \mathcal{P}_i\}$. We sometimes write

$$\{P_{k,1}(\widetilde{x}_{k,1}) =_{\alpha_k} \varphi_{k,1}, \ldots, P_{k,\ell_k}(\widetilde{x}_{k,\ell_k}) =_{\alpha_k} \varphi_{k,\ell_k}\}; \cdots;$$
$$\{P_{1,1}(\widetilde{x}_{1,1}) =_{\alpha_1} \varphi_{1,1}, \ldots, P_{1,\ell_1}(\widetilde{x}_{1,\ell_1}) =_{\alpha_1} \varphi_{1,\ell_1}\}$$

for $(\mathcal{E}, \mathcal{H})$ where $\mathcal{E} = \{P_{i,j}(\widetilde{x}_{i,j}) =_{\alpha_i} \varphi_{i,j} \mid i \in \{1, \ldots, k\}, j \in \{1, \ldots, \ell_i\}\}$ and $\mathcal{H} = (\{P_{k,1}, \ldots, P_{k,\ell_k}\}, \alpha_k); \cdots; (\{P_{1,1}, \ldots, P_{1,\ell_1}\}, \alpha_1)$. When $P(\widetilde{x}) = \varphi \in \mathcal{E}$, we write $\mathtt{ar}_{\mathcal{E}}(P)$ for the length $|\widetilde{x}|$ of the sequence $\widetilde{x}$; we often omit the subscript when $\mathcal{E}$ is clear from context.

Bound (integer) variables in a formula are defined as usual. Integer variables $\{\widetilde{x}\}$ are bound in an equation $P(\widetilde{x}) = \varphi$. We assume that a given HES is closed: free predicate variables in each formula are defined in the HES.

Intuitively, $(\{P_{i,1}, \ldots, P_{i,\ell_i}\}, \alpha_i)$ where $\alpha_i = \mu$ ($\alpha_i = \nu$, resp.) means that $P_{i,1}, \ldots, P_{i,\ell_i}$ are the *least* (*greatest*, resp.) predicates that satisfy the corresponding equations. In $\mathcal{H} = (\{P_{k,1}, \ldots, P_{k,\ell_k}\}, \alpha_k); \cdots; (\{P_{1,1}, \ldots, P_{1,\ell_1}\}, \alpha_1)$, the predicates in $\{P_{k,1}, \ldots, P_{k,\ell_k}\}$ ($\{P_{1,1}, \ldots, P_{1,\ell_1}\}$, resp.) are bound in the outermost (innermost, resp.) position.

Example 1. Consider the HES[5] $\Phi = (\mathcal{E}, \mathcal{H})$ with $\mathcal{H} = (\{P_2\}, \nu); (\{P_1\}, \mu)$ and

$$\mathcal{E} = \{P_2(x) = P_2(x+1) \wedge P_1(x,0), P_1(x,y) = (y = x \vee P_1(x, y+1))\}.$$

Since $P_1(x,y)$ can be expanded to:

$$P_1(x,y) \equiv y = x \vee P_1(x,y+1) \equiv y = x \vee y+1 = x \vee P_1(x,y+2) \equiv \cdots,$$

$P_1(x,y)$ is equivalent to $x \geq y$. Thus, $P_2(x)$ is equivalent to:

$$P_2(x+1) \wedge x \geq 0 \equiv P_2(x+2) \wedge x+1 \geq 0 \wedge x \geq 0 \equiv \cdots.$$

Therefore, $P_2(x)$ is equivalent to $x \geq 0$. □

Remark 1. We do not have the negation operator as a primitive, but it can be expressed by using de Morgan duality [29]. The quantifiers $\forall, \exists$ could also be removed: A formula $\exists x.\varphi$ with the free variables $\widetilde{y}$ can be expressed by $P(0, \widetilde{y})$ where P is defined by $P(x, \widetilde{y}) =_\mu \varphi \vee P(x+1, \widetilde{y}) \vee P(x-1, \widetilde{y})$; similarly for $\forall x.\varphi$.

2.2 Semantics

We now define the formal semantics of HES. Let $\mathbf{Z}$ and $\mathbf{B} = \{\mathtt{tt}, \mathtt{ff}\}$ be the sets of integers and Booleans, respectively. We consider the partial order $\mathtt{ff} \sqsubseteq \mathtt{tt}$ on $\mathbf{B}$, and write $\sqcup$ ($\sqcap$, resp.) for the least upper (greatest lower, resp.) bound with respect to $\sqsubseteq$. Given Φ and $\mathcal{P} \subseteq \mathcal{P}_\Phi$, we write $\Gamma_\mathcal{P}$ for the set of maps ρ such that $dom(\rho) = \mathcal{P}$ and $\rho(P) \in \mathbf{Z}^{\mathtt{ar}(P)} \to \mathbf{B}$ for $P \in dom(\rho)$. $(\Gamma_\mathcal{P}, \sqsubseteq_{\Gamma_\mathcal{P}})$ forms a complete lattice, where $\Gamma_\mathcal{P}$ is the pointwise ordering on the elements of $\Gamma_\mathcal{P}$.

Given a map ρ such that $dom(\rho) = \mathcal{P} \cup \mathcal{X}$, where $\mathcal{X}$ is a finite subset of integer variables, and $\rho(P) \in \mathbf{Z}^{\mathtt{ar}(P)} \to \mathbf{B}$ for $P \in \mathcal{P}$ and $\rho(x) \in \mathbf{Z}$ for $x \in \mathcal{X}$, the semantics $[\![\varphi]\!]_\rho \in \mathbf{B}$ of a formula φ is defined by:

$$[\![a_1 \geq a_2]\!]_\rho = \begin{cases} \mathtt{tt} \text{ if } [\![a_1]\!]_\rho \geq [\![a_2]\!]_\rho \\ \mathtt{ff} \text{ otherwise} \end{cases} \qquad [\![P(a_1, \ldots, a_m)]\!]_\rho = \rho(P)([\![a_1]\!]_\rho, \ldots, [\![a_m]\!]_\rho)$$

$$[\![\varphi_1 \vee \varphi_2]\!]_\rho = [\![\varphi_1]\!]_\rho \sqcup [\![\varphi_1]\!]_\rho \qquad [\![\varphi_1 \wedge \varphi_2]\!]_\rho = [\![\varphi_1]\!]_\rho \sqcap [\![\varphi_1]\!]_\rho$$

$$[\![\exists x.\varphi]\!]_\rho = \bigsqcup_{z \in \mathbf{Z}} [\![\varphi]\!]_{\rho\{x \mapsto z\}} \qquad [\![\forall x.\varphi]\!]_\rho = \bigsqcap_{z \in \mathbf{Z}} [\![\varphi]\!]_{\rho\{x \mapsto z\}}$$

$$[\![x]\!]_\rho = \rho(x) \qquad [\![n]\!]_\rho = n \qquad [\![a_1 \,\mathbf{op}\, a_2]\!]_\rho = [\![a_1]\!]_\rho [\![\mathbf{op}]\!] [\![a_2]\!]_\rho.$$

Here, $[\![\mathbf{op}]\!]$ denotes the binary function on integers represented by $\mathbf{op}$.

Given an HES Φ, $\mathcal{P}_i$ and $\rho \in \Gamma_{\mathcal{P}_\Phi}$, the semantics $[\![\mathcal{E}_i]\!]_\rho \in \Gamma_{\mathcal{P}_i}$ is defined by:

$$[\![\mathcal{E}_i]\!]_\rho = \{P \mapsto \lambda \widetilde{z} \in \mathbf{Z}^{\mathtt{ar}(P)}.[\![\varphi_P]\!]_{\rho\{\widetilde{x} \mapsto \widetilde{z}\}} \mid P(\widetilde{x}) = \varphi_P \in \mathcal{E}_i\}.$$

We are now ready to define the semantics of HES. By abuse of notation, we write Γ_i, $\Gamma_{\geq i}$, and $\Gamma_{\leq i}$ for $\Gamma_{\mathcal{P}_i}$, $\Gamma_{\mathcal{P}_{\geq i}}$, and $\Gamma_{\mathcal{P}_{\leq i}}$ respectively. The semantics

[5] We remark that, for those who are familiar with fixpoint logics, $P_2(x)$ can be written as $\nu P_2.\lambda x.P_2(x+1) \wedge (\mu P_1.\lambda y.y = x \vee P_1(y+1))\,0$ in the ordinary syntax of Mu-Arithmetic [11] or HFL [39].

$[\![\Phi]\!]_i \in \Gamma_{\geq i+1} \to \Gamma_i$ and $[\![\Phi]\!]_{\leq i} \in \Gamma_{\geq i+1} \to \Gamma_{\leq i}$ are defined by induction on i, as follows.

$$\begin{aligned}
&[\![\Phi]\!]_0 = [\![\Phi]\!]_{\leq 0} = \lambda\rho \in \Gamma_{\geq 1}.\emptyset \\
&[\![\Phi]\!]_i = \lambda\rho \in \Gamma_{\geq i+1}.\mathbf{FP}_{\alpha_i}^{\Gamma_i}(\lambda\rho' \in \Gamma_i.[\![\mathcal{E}_i]\!]_{\rho\cup\rho'\cup[\![\Phi]\!]_{\leq i-1}(\rho\cup\rho')}) \quad \text{(for } i > 0\text{)} \\
&[\![\Phi]\!]_{\leq i} = \lambda\rho \in \Gamma_{\geq i+1}.[\![\Phi]\!]_i(\rho) \cup [\![\Phi]\!]_{\leq i-1}(\rho \cup [\![\Phi]\!]_i(\rho)) \quad \text{(for } i > 0\text{)}
\end{aligned}$$

Here, $\mathbf{FP}_\mu^\Gamma, \mathbf{FP}_\nu^\Gamma \in (\Gamma \to \Gamma) \to \Gamma$ (the superscript Γ is often omitted) are the least and greatest fixpoint operators defined by:

$$\mathbf{FP}_\mu^\Gamma(F) = \bigsqcap_\Gamma\{f \in \Gamma \mid f \sqsupseteq_\Gamma F(f)\} \qquad \mathbf{FP}_\nu^\Gamma(F) = \bigsqcup_\Gamma\{f \in \Gamma \mid f \sqsubseteq_\Gamma F(f)\}.$$

Note that the semantics $[\![\Phi]\!]_i$ of the predicates in $\mathcal{P}_i$ is parameterized by the semantics of predicates of higher levels (as indicated by $\lambda\rho \in \Gamma_{\geq i+1}.\cdots$). To evaluate $[\![\mathcal{E}_i]\!]$ in the definition of $[\![\Phi]\!]_i$, we need an environment on all the predicate variables; ρ, ρ', and $[\![\Phi]\!]_{\leq i-1}(\rho \cup \rho')$ respectively provide the environment on the predicates of higher levels, the current level i, and lower levels. We write $[\![\Phi]\!]$ for $[\![\Phi]\!]_{\leq k}(\emptyset)$ (where k is the highest level) and write $\Phi \models \varphi$ if $[\![\varphi]\!]_{[\![\Phi]\!]} = \mathtt{tt}$.

Example 2. Recall Example 1, with $\mathcal{P}_1 = \{P_1\}$ and $\mathcal{P}_2 = \{P_2\}$.

$$\begin{aligned}
[\![\Phi]\!]_1 &= \lambda\rho \in \Gamma_{\geq 2}.\mathbf{FP}_\mu(\lambda\rho' \in \Gamma_1.\{P_1 \mapsto \lambda(x,y).y = x \vee \rho'(P_1)(x,y+1)\}) \\
&= \lambda\rho \in \Gamma_{\geq 2}.\{P_1 \mapsto \lambda(x,y).x \geq y\}. \\
[\![\Phi]\!]_{\leq 1} &= \lambda\rho \in \Gamma_{\geq 2}.[\![\Phi]\!]_1(\rho) \cup [\![\Phi]\!]_{\leq 0}(\rho \cup [\![\Phi]\!]_1(\rho)) = [\![\Phi]\!]_1. \\
[\![\Phi]\!]_2 &= \lambda\rho \in \Gamma_{\geq 3}. \\
&\qquad \mathbf{FP}_\nu(\lambda\rho' \in \Gamma_2.\{P_2 \mapsto \lambda x.\rho'(P_2)(x+1) \wedge [\![\Phi]\!]_{\leq 1}(\rho \cup \rho')(P_1)(x,0)\}) \\
&= \lambda\rho \in \{\emptyset\}.\mathbf{FP}_\nu(\lambda\rho' \in \Gamma_2.\{P_2 \mapsto \lambda x.\rho'(P_2)(x+1) \wedge x \geq 0\}) \\
&= \lambda\rho \in \{\emptyset\}.\{P_2 \mapsto \lambda x.x \geq 0\}. \\
[\![\Phi]\!]_{\leq 2} &= \lambda\rho \in \{\emptyset\}.\{P_2 \mapsto \lambda x.x \geq 0, P_1 \mapsto \lambda(x,y).x \geq y\}.
\end{aligned}$$

Thus, we have $[\![\Phi]\!] = \{P_2 \mapsto \lambda x.x \geq 0, P_1 \mapsto \lambda(x,y).x \geq y\}$, hence $\Phi \models P_2(0)$. □

Example 3. To understand the importance of the order of equations, let us consider $\Phi_1 = (\mathcal{E}, \mathcal{H}_1)$ and $\Phi_2 = (\mathcal{E}, \mathcal{H}_2)$, where:

$$\begin{aligned}
&\mathcal{E} = \{X = X \wedge Y, Y = X \vee Y\} \\
&\mathcal{H}_1 = (\{X\},\nu);(\{Y\},\mu) \qquad \mathcal{H}_2 = (\{Y\},\mu);(\{X\},\nu).
\end{aligned}$$

Note that Φ_2 is obtained from Φ_1 by just swapping the order of X and Y. Yet, their semantics are completely different: $[\![\Phi_1]\!] = \{X \mapsto \mathtt{tt}, Y \mapsto \mathtt{tt}\}$ but $[\![\Phi_2]\!] = \{X \mapsto \mathtt{ff}, Y \mapsto \mathtt{ff}\}$. To see this, for Φ_1, we have:

$$\begin{aligned}
[\![\Phi_1]\!]_1 &= \lambda\rho \in \Gamma_{\geq 2}.\mathbf{FP}_\mu(\lambda\rho' \in \Gamma_1.[\![Y = X \vee Y]\!]_{\rho\cup\rho'}) = \lambda\rho \in \Gamma_{\geq 2}.\{Y \mapsto \rho(X)\} \\
[\![\Phi_1]\!]_2 &= \lambda\rho \in \{\emptyset\}.\mathbf{FP}_\nu(\lambda\rho' \in \Gamma_2.[\![X = X \wedge Y]\!]_{\rho'\cup\{Y\mapsto\rho(X)\}}) \\
&= \lambda\rho \in \{\emptyset\}.\{X \mapsto \mathtt{tt}\}.
\end{aligned}$$

In contrast, for Φ_2, we have:

$$\begin{aligned}
[\![\Phi_2]\!]_1 &= \lambda\rho \in \Gamma_{\geq 2}.\mathbf{FP}_\nu(\lambda\rho' \in \Gamma_1.[\![X = X \wedge Y]\!]_{\rho\cup\rho'}) = \lambda\rho \in \Gamma_{\geq 2}.\{X \mapsto \rho(Y)\} \\
[\![\Phi_2]\!]_2 &= \lambda\rho \in \{\emptyset\}.\mathbf{FP}_\mu(\lambda\rho' \in \Gamma_2.[\![Y = X \vee Y]\!]_{\rho'\cup\{X\mapsto\rho(Y)\}}) \\
&= \lambda\rho \in \{\emptyset\}.\{Y \mapsto \mathtt{ff}\}.
\end{aligned}$$

□

2.3 Relationship with Other Logics

We comment on the relationship between our logic and other logics used in the context of program verification. As indicated already, our fixpoint logic in the form of HES is essentially equi-expressive as Bradfield's original Mu-Arithmetic [11]. Any formula of HES can be translated to a formula of the original Mu-Arithmetic in the same way as the translation from HES for higher-order fixpoint logic to HFL formulas [25].

Our Mu-Arithmetic can be considered a restriction of $\mathrm{HFL}_{\mathbf{Z}}$ [26] (which is an extension of HFL [39] with integers), obtained by (i) restricting predicates to those on integers, and (ii) removing modal operators.

HES can also be considered an extension of Constrained Horn Clauses (CHC) [6], obtained by allowing fixpoint alternations. In fact, the satisfiability problem of CHC (i.e., whether there is a substitution for the predicate variables $P_1, \ldots, P_n$ that makes all the clauses valid):

$$P_1(\widetilde{x}_1) \Leftarrow \varphi_1 \qquad \cdots \qquad P_n(\widetilde{x}_n) \Leftarrow \varphi_n \qquad \mathtt{ff} \Leftarrow P_1(\widetilde{x}_1)$$

(where we allow disjunctions in $\varphi_1, \ldots, \varphi_n$, and assume that $P_1, \ldots, P_n$ are mutually distinct) is equivalent to the validity of $\forall \widetilde{x}_1.\overline{P}_1(\widetilde{x}_1)$, where $\overline{P}_i$ is defined by HES: $(\{\overline{P}_1(\widetilde{x}_1) = \overline{\varphi}_1, \ldots, \overline{P}_n(\widetilde{x}_n) = \overline{\varphi}_n\},\ (\{\overline{P}_1, \ldots, \overline{P}_n\}, \nu))$. Here, $\overline{\varphi}_i$ is the de Morgan dual of φ_i, and $\overline{P}_i$ intuitively represents the negation of P_i.

Conversely, the validity checking problem for any HES without μ or $\exists$ can be transformed to the satisfiability problem for CHC, by just reversing the above transformation. In fact, let Φ be an HES of the form

$$(\{P_1(\widetilde{x}_1) = \forall \widetilde{y}_1.\varphi_1, \ldots, P_n(\widetilde{x}_n) = \forall \widetilde{y}_n.\varphi_n\},\ (\{P_1, \ldots, P_n\}, \nu)),$$

where $\varphi_1, \ldots, \varphi_n$ are quantifier-free formulas. Then, $\Phi \models \forall \widetilde{x}.P_i(\widetilde{x})$ if and only if the followings are satisfiable:

$$\overline{P}_1(\widetilde{x}_1) \Leftarrow \exists \widetilde{y}_1.\overline{\varphi}_1 \qquad \cdots \qquad \overline{P}_n(\widetilde{x}_n) \Leftarrow \exists \widetilde{y}_n.\overline{\varphi}_n \qquad \mathtt{ff} \Leftarrow \exists \widetilde{x}.\overline{P}_i(\widetilde{x}).$$

Since $\overline{P}_i(\widetilde{x}_i) \Leftarrow \exists \widetilde{y}_i.\overline{\varphi}_i$ is equivalent to $\forall \widetilde{y}_i.(\overline{P}_i(\widetilde{x}_i) \Leftarrow \overline{\varphi}_i)$ (assuming that $\widetilde{x}_i \cap \widetilde{y}_i = \emptyset$), one can transform the conditions above to CHC.

3 From Temporal Property Verification to First-Order Fixpoint Logic

This section discusses applications of the fixpoint logic to temporal verification of programs. As we mentioned in Section 1, Watanabe et al. [40] have shown that temporal verification of higher-order recursive programs can be reduced in a sound and complete manner to validity checking of *higher-order* fixpoint logic formulas. For while-programs (i.e., imperative programs without recursion), their translations actually produce formulas within our fixpoint logic. Thus, by combining their translations with the procedures for our fixpoint logic given in Section 4, we obtain an automated temporal verification method for while-programs,

which can deal with arbitrary temporal properties that can be expressed in modal μ-calculus. The reduction of Watanabe et al. [40] is, however, indirect: one has to first transform a while-program to a tree-generating grammar $\mathrm{HORS}_{\mathbf{Z}}$ that generates a computation tree of the program, and a temporal property to a tree automaton. We thus present a direct reduction from modal μ-calculus model checking of imperative programs to validity checking of Mu-Arithmetic formulas in Section 3.1 below.

For linear-time temporal properties, we can also deal with first-order programs with arbitrary recursion (not just while-loops). More precisely, given a first-order (possibly non-deterministic) recursive program D (that contains special primitives called *events*) and a Büchi automaton $\mathcal{A}$, one can construct an HES $\Phi_{D,\mathcal{A}}$, such that $\Phi_{D,\mathcal{A}} \models \mathbf{main}_{q_I,\mathtt{tt}}()$ (where q_I is the initial state of $\mathcal{A}$) holds, if and only if some (infinite) event sequence generated by D is accepted by $\mathcal{A}$. We formalize the reduction in Section 3.2, which is one of the main contributions of the present paper.

3.1 Modal μ-Calculus Model Checking of Imperative Programs

We model an imperative program (without recursion) as a tuple $\mathbf{P} = (\mathbf{PC}, \mathbf{Vars}, \mathbf{Code})$, where $\mathbf{PC}$ is a finite set consisting of non-negative integers (which intuitively represent program counters), $\mathbf{Vars}$ is a finite set of variables, and $\mathbf{Code}$ is a map from $\mathbf{PC}$ to the set $\mathcal{I}_{\mathbf{Vars}}$ of instructions, consisting of:

- $x := a; \mathbf{goto}\ i$: update the value of $x \in \mathbf{Vars}$ to that of a, and then go to $i \in \mathbf{PC}$.
- $x := *; \mathbf{goto}\ i$: update the value of $x \in \mathbf{Vars}$ to an arbitrary integer in a non-deterministic manner, and then go to $i \in \mathbf{PC}$.
- $\mathbf{if}\ a_1 \geq a_2\ \mathbf{then\ goto}\ i\ \mathbf{else\ goto}\ j$: go to $i \in \mathbf{PC}$ if $a_1 \geq a_2$ and $j \in \mathbf{PC}$ otherwise.
- $\mathbf{if}\ *\ \mathbf{then\ goto}\ i\ \mathbf{else\ goto}\ j$: non-deterministically go to i or j.

Here, a ranges over the set of arithmetic expressions, like the meta-variable a in Section 2.

A program $\mathbf{P} = (\mathbf{PC}, \mathbf{Vars}, \mathbf{Code})$, with $\mathbf{Vars} = \{x_1, \ldots, x_n\}$ can be viewed as a Kripke structure $K_{\mathbf{P}} = (\mathbf{AP}, S, s_0, \longrightarrow, L)$, where:

- $\mathbf{AP}$ is a set of constraints on $\mathbf{Vars}$ (such as $x_1 \geq 0$),
- the set S of states is $\mathbf{PC} \times (\mathbf{Vars} \to \mathbf{Z})$,
- the initial state $s_0 \in S$ is $(0, \{x_1 \mapsto 0, \ldots, x_n \mapsto 0\})$,
- the labeling function $L \in S \to \mathbf{AP}$ is given by: $L(i, \sigma) = \{p \in \mathbf{AP} \mid\models \sigma(p)\}$. Here, $\sigma(p)$ is the closed formula obtained by replacing each variable x_i in p with $\sigma(x_i)$, and $\models \sigma(p)$ means that the resulting formula evaluates to true, and
- the transition relation $\longrightarrow$ is defined by the following rules.

$$\frac{\mathbf{Code}(i) = (x_j := a; \mathbf{goto}\ k)}{(i, \sigma) \longrightarrow (k, \sigma\{x_j \mapsto \mathbf{val}(\sigma(a))\})}$$

$$\frac{\mathbf{Code}(i) = (x_j := *; \mathbf{goto}\ k)}{(i,\sigma) \longrightarrow (k, \sigma\{x_j \mapsto m\})}$$

$$\frac{\mathbf{Code}(i) = \mathbf{if}\ a_1 \geq a_2\ \mathbf{then\ goto}\ j\ \mathbf{else\ goto}\ k \qquad \mathbf{val}(\sigma(a_1)) \geq \mathbf{val}(\sigma(a_2))}{(i,\sigma) \longrightarrow (j,\sigma)}$$

$$\frac{\mathbf{Code}(i) = \mathbf{if}\ a_1 \geq a_2\ \mathbf{then\ goto}\ j\ \mathbf{else\ goto}\ k \qquad \mathbf{val}(\sigma(a_1)) < \mathbf{val}(\sigma(a_2))}{(i,\sigma) \longrightarrow (k,\sigma)}$$

$$\frac{\mathbf{Code}(i) = \mathbf{if}\ *\ \mathbf{then\ goto}\ j_1\ \mathbf{else\ goto}\ j_2 \qquad k \in \{1,2\}}{(i,\sigma) \longrightarrow (j_k,\sigma)}$$

We represent modal μ-calculus formulas in the form of hierarchical equation systems, following [36]. We call them *hierarchical modal equation systems (HMES)*, to distinguish them from the HES for Mu-Arithmetic introduced in Section 2.

The set of (fixpoint-free) modal formulas, ranged over by ψ, is defined by:

$$\psi ::= p \mid X \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \diamond\psi \mid \Box\psi$$
$$a ::= n \mid x \mid a_1\ \mathbf{op}\ a_2.$$

A *hierarchical modal equation system* (HMES) Ξ is a pair $(\mathbf{E}, \mathbf{H})$, where $\mathbf{E}$ is a set of equations for the form:

$$\{X_1 = \psi_1, \cdots, X_m = \psi_n\},$$

where $\mathbf{H}$ is a sequence $(\mathcal{P}_k, \alpha_k); \cdots; (\mathcal{P}_1, \alpha_1)$, where $\mathcal{P}_k, \ldots, \mathcal{P}_1$ are mutually disjoint sets of variables such that $\mathcal{P}_k \cup \cdots \cup \mathcal{P}_1 = \{X_1, \ldots, X_m\}$, and $\alpha_i \in \{\mu, \nu\}$. We write $\mathcal{P}_{\geq i}$ and $\mathcal{P}_{\leq i}$ for $\bigcup_{j \geq i} \mathcal{P}_j$ and $\bigcup_{j \leq i} \mathcal{P}_j$ respectively.

The semantics of HMES is defined in a way analogous to that of HES in Section 2. First, given a function ρ which maps each fixpoint variable to a set of states, we define $[\![\psi]\!]_\rho$ by

$$[\![a_1 \geq a_2]\!]_\rho = \{(i,\sigma) \mid \mathbf{val}(\sigma(a_1)) \geq \mathbf{val}(\sigma(a_2))\}$$
$$[\![X]\!]_\rho = \rho(X) \qquad [\![\psi_1 \vee \psi_2]\!]_\rho = [\![\psi_1]\!]_\rho \cup [\![\psi_2]\!]_\rho \qquad [\![\psi_1 \wedge \psi_2]\!]_\rho = [\![\psi_1]\!]_\rho \cap [\![\psi_2]\!]_\rho$$
$$[\![\diamond\psi]\!]_\rho = \{(i,\sigma) \mid (i,\sigma) \longrightarrow (j,\sigma') \text{ for some } (j,\sigma') \in [\![\psi]\!]_\rho\}$$
$$[\![\Box\psi]\!]_\rho = \{(i,\sigma) \mid (j,\sigma') \in [\![\psi]\!]_\rho \text{ for every } (j,\sigma') \text{ such that } (i,\sigma) \longrightarrow (j,\sigma')\}$$

We write $\mathbf{E}_i$ for the subset of equations: $\{X_j = \psi_j \mid X_j \in \mathcal{P}_i\}$. The semantics $[\![\mathbf{E}_i]\!]_\rho$ is defined by:

$$[\![\mathbf{E}_i]\!]_\rho = \{X \mapsto [\![\psi]\!]_\rho \mid X = \psi \in \mathbf{E}_i\}.$$

The semantics $[\![\Xi]\!]$ of HMES Ξ is $[\![\Xi]\!]_{\leq k}$, where

$$[\![\Xi]\!]_0 = [\![\Xi]\!]_{\leq 0} = \lambda\rho \in \Theta_{\geq 1}.\emptyset$$
$$[\![\Xi]\!]_i = \lambda\rho \in \Theta_{\geq i+1}.\mathbf{FP}^{\Theta_i}_{\alpha_i}(\lambda\rho' \in \Theta_i.[\![\mathbf{E}_i]\!]_{\rho \cup \rho' \cup [\![\Xi]\!]_{\leq i-1}(\rho \cup \rho')}) \text{ (for } i > 0)$$
$$[\![\Xi]\!]_{\leq i} = \lambda\rho \in \Theta_{\geq i+1}.[\![\Xi]\!]_i(\rho) \cup [\![\Xi]\!]_{\leq i-1}(\rho \cup [\![\Xi]\!]_i(\rho)) \qquad (\text{for } i > 0).$$

Here, $\Theta_i = \mathcal{P}_i \to 2^{2^{\mathbf{PC}\times(\mathbf{Vars}\to Z)}}$ and $\Theta_{\geq i} = \mathcal{P}_{\geq i} \to 2^{2^{\mathbf{PC}\times(\mathbf{Vars}\to Z)}}$.

Given a program $\mathbf{P}$ with variables $\{x_1, \ldots, x_n\}$ and an HMES Ξ with $X \in \mathcal{P}_\Xi$, we say that a program $\mathbf{P}$ satisfies (Ξ, X), written $\mathbf{P} \models (\Xi, X)$, if the initial state $(0, \{x_1 \mapsto 0, \ldots, x_n \mapsto 0\})$ belongs to $[\![\Xi]\!](X)$. The goal of verification is to check whether $\mathbf{P} \models (\Xi, X)$ holds.

We now reduce the problem of checking whether $\mathbf{P} \models (\Xi, X)$ to the validity checking problem for Mu-Arithmetic. For the convenience of presenting the reduction, we assume without loss of generality that the righthand side of each equation in HMES is restricted to the following syntax.

$$\psi ::= a_1 \geq a_2 \mid X \mid X_1 \vee X_2 \mid X_1 \wedge X_2 \mid \diamond X \mid \Box X.$$

Let $\mathbf{Vars} = \{x_1, \ldots, x_n\}$ and $\widetilde{x}$ be the sequence $x_1, \ldots, x_n$. For each equation $X = \psi$ and $i \in \mathbf{PC}$, we define the equation $[\![X = \psi]\!]_i$ of Mu-Arithmetic by:

$$[\![X = (a_1 \geq a_2)]\!]_i = (X^{(i)}(\widetilde{x}) = a_1 \geq a_2) \qquad [\![X = X_1]\!]_i = (X^{(i)}(\widetilde{x}) = X_1^{(i)})$$
$$[\![X = X_1 \vee X_2]\!]_i = (X^{(i)}(\widetilde{x}) = X_1^{(i)}(\widetilde{x}) \vee X_2^{(i)}(\widetilde{x}))$$
$$[\![X = X_1 \wedge X_2]\!]_i = (X^{(i)}(\widetilde{x}) = X_1^{(i)}(\widetilde{x}) \wedge X_2^{(i)}(\widetilde{x}))$$

$$[\![X = \diamond X_1]\!]_i = \begin{cases} X^{(i)}(\widetilde{x}) = X_1^{(k)}(x_1, \ldots, x_{j-1}, a, x_{j+1}, \ldots, x_n) \\ \qquad\qquad \text{if } \mathbf{Code}(i) = x_j := a; \mathbf{goto}\ k \\ X^{(i)}(\widetilde{x}) = \exists m. X_1^{(k)}(x_1, \ldots, x_{j-1}, m, x_{j+1}, \ldots, x_n) \\ \qquad\qquad \text{if } \mathbf{Code}(i) = x_j := *; \mathbf{goto}\ k \\ X^{(i)}(\widetilde{x}) = (a_1 \geq a_2 \wedge X_1^{(j)}(\widetilde{x})) \vee (a_2 \geq a_1 + 1 \wedge X_1^{(k)}(\widetilde{x})) \\ \qquad \text{if } \mathbf{Code}(i) = \mathbf{if}\ a_1 \geq a_2\ \mathbf{then\ goto}\ j\ \mathbf{else\ goto}\ k \\ X^{(i)}(\widetilde{x}) = X_1^{(j)}(\widetilde{x}) \vee X_1^{(k)}(\widetilde{x}) \\ \qquad\qquad \text{if } \mathbf{Code}(i) = \mathbf{if}\ *\ \mathbf{then\ goto}\ j\ \mathbf{else\ goto}\ k \end{cases}$$

$$[\![X = \Box X_1]\!]_i = \begin{cases} X^{(i)}(\widetilde{x}) = X_1^{(k)}(x_1, \ldots, x_{j-1}, a, x_{j+1}, \ldots, x_n) \\ \qquad\qquad \text{if } \mathbf{Code}(i) = x_j := a; \mathbf{goto}\ k \\ X^{(i)}(\widetilde{x}) = \forall m. X_1^{(k)}(x_1, \ldots, x_{j-1}, m, x_{j+1}, \ldots, x_n) \\ \qquad\qquad \text{if } \mathbf{Code}(i) = x_j := *; \mathbf{goto}\ k \\ X^{(i)}(\widetilde{x}) = (a_1 \geq a_2 \wedge X_1^{(j)}(\widetilde{x})) \vee (a_2 \geq a_1 + 1 \wedge X_1^{(k)}(\widetilde{x})) \\ \qquad \text{if } \mathbf{Code}(i) = \mathbf{if}\ a_1 \geq a_2\ \mathbf{then\ goto}\ j\ \mathbf{else\ goto}\ k \\ X^{(i)}(\widetilde{x}) = X_1^{(j)}(\widetilde{x}) \wedge X_1^{(k)}(\widetilde{x}) \\ \qquad\qquad \text{if } \mathbf{Code}(i) = \mathbf{if}\ *\ \mathbf{then\ goto}\ j\ \mathbf{else\ goto}\ k. \end{cases}$$

Given $\Xi = (\mathbf{E}, \mathbf{H})$ with $\mathbf{H} = (\mathcal{P}_1, \alpha_1); \cdots; (\mathcal{P}_\ell, \alpha_\ell)$, and $\mathbf{P} = (\mathbf{PC}, \mathbf{Vars}, \mathbf{Code})$, we define HES $\Phi_{\Xi,\mathbf{P}}$ as $(\mathcal{E}_{\Xi,\mathbf{P}}, \mathcal{H}_{\Xi,\mathbf{P}})$, where:

$$\mathcal{E}_{\Xi,\mathbf{P}} = \{[\![X = \psi]\!]_i \mid (X = \psi) \in \mathbf{E}, i \in \mathbf{PC}\},$$

and

$$\mathcal{H}_{\Xi,\mathbf{P}} = (\{X^{(i)} \mid X \in \mathcal{P}_1, i \in \mathbf{PC}\}, \alpha_1); \cdots; (\{X^{(i)} \mid X \in \mathcal{P}_\ell, i \in \mathbf{PC}\}, \alpha_\ell).$$

By the translation, it is not difficult to observe that $(j, \{x_1 \mapsto m_1, \ldots, x_n \mapsto m_n\}) \in [\![\Xi]\!](X_i)$ if and only if $[\![\Phi_{\Xi,\mathbf{P}}]\!](X_i^{(j)})(m_1, \ldots, m_n) = \mathtt{tt}$. Thus, $\mathbf{P} \models (\Xi, X_1)$ if and only if $\mathcal{H}_{\Xi,\mathbf{P}} \models X_1^{(0)}(0, \ldots, 0)$.

Example 4. Consider the program $\mathbf{P}_0 = (\{0,1\}, \{x,y\}, \mathbf{Code}_0)$, where:

$$\mathbf{Code}_0 = \{0 \mapsto (x := x-1; \mathbf{goto}\ 1),\ 1 \mapsto (y := y+1; \mathbf{goto}\ 0)\}.$$

The modal μ-calculus formula $\nu X.(x+y \geq 0 \wedge \Diamond\Diamond X)$ expresses the property "there exists an execution sequence in which $x+y \geq 0$ holds after any even number of steps." This is a property that cannot be expressed in CTL*. The corresponding HMES Ξ_0 is $(\mathbf{E}_0, \mathbf{H}_0)$ where

$$\mathbf{E}_0 = \{X = x+y \geq 0 \wedge \Diamond Y, Y = \Diamond X\} \qquad \mathbf{H}_0 = (\{X,Y\}, \nu).$$

By the translation above, we obtain $\Phi_{\Xi_0,\mathbf{P}_0} = (\mathcal{E}_0, \mathcal{H}_0)$ where $\mathcal{H}_0 = (\{X^{(0)}, X^{(1)}, Y^{(0)}, Y^{(1)}\}, \nu)$ and $\mathcal{E}_0$ consists of:

$$X^{(0)}(x,y) = x+y \geq 0 \wedge Y^{(1)}(x-1,y) \quad X^{(1)}(x,y) = x+y \geq 0 \wedge Y^{(0)}(x,y+1)$$
$$Y^{(0)}(x,y) = X^{(1)}(x-1,y) \qquad Y^{(1)}(x,y) = X^{(0)}(x,y+1)$$

□

Example 5. Consider the program $\mathbf{P}_1 = (\{0,1,2\}, \{x\}, \mathbf{Code}_1)$, where $\mathbf{Code}_1$ consists of:

$$0 \mapsto (x := *; \mathbf{goto}\ 1),$$
$$1 \mapsto (\mathbf{if}\ 0 \geq x\ \mathbf{then\ goto}\ 0\ \mathbf{else\ goto}\ 2),$$
$$2 \mapsto (x := x-1; \mathbf{goto}\ 1).$$

Let us consider the μ-calculus formula

$$\nu X.\mu Y.\Box Y \vee (0 \geq x \wedge (\mu Z.\Diamond Z \vee (x \geq 1 \wedge X))),$$

which corresponds to the following infinite CTL formula:

$$\mathtt{AF}(0 \geq x \wedge \mathtt{EF}(x \geq 1 \wedge \mathtt{AF}(0 \geq x \wedge \mathtt{EF}(x \geq 1 \wedge \cdots)))).$$

It is expressed as $\Xi_1 = (\mathbf{E}_1, \mathbf{H}_1)$ where:

$$\mathbf{E}_1 = \{X = Y, Y = Y_1 \vee Y_2, Y_1 = \Box Y, Y_2 = Y_{21} \vee Z, Y_{21} = 0 \geq x,$$
$$Z = Z_1 \vee Z_2, Z_1 = \Diamond Z, Z_2 = Z_{21} \wedge X, Z_{21} = x \geq 1\}$$
$$\mathbf{H}_1 = (\{X\}, \nu); (\{Y, Y_1, Y_2, Y_{21}, Z, Z_1, Z_2, Z_{21}\}, \mu).$$

Our translation yields $\Phi_{\Xi_1,\mathbf{P}_1} = (\mathcal{E}_1, \mathcal{H}_1)$ where

$$\begin{aligned}
\mathcal{E}_1 = \ & \{X^{(i)}(x) = Y^{(i)}(x) \mid i \in \{0,1,2\}\} \\
& \cup \{Y^{(i)}(x) = Y_1^{(i)}(x) \vee Y_2^{(i)}(x) \mid i \in \{0,1,2\}\} \\
& \cup \{Y_1^{(0)}(x) = \forall x.Y^{(1)}(x), Y_1^{(2)}(x) = Y^{(1)}(x-1), \\
& \quad Y_1^{(1)}(x) = (0 \geq x \wedge Y^{(0)}(x)) \vee (x \geq 1 \wedge Y^{(2)}(x))\} \\
& \cup \{Y_2^{(i)}(x) = Y_{21}^{(i)}(x) \vee Z^{(i)}(x) \mid i \in \{0,1,2\}\} \\
& \cup \{Y_{21}^{(i)}(x) = 0 \geq x \mid i \in \{0,1,2\}\} \\
& \cup \{Z^{(i)}(x) = Z_1^{(i)}(x) \vee Z_2^{(i)}(x) \mid i \in \{0,1,2\}\} \\
& \cup \{Z_1^{(0)}(x) = \exists x.Z^{(1)}(x), Z_1^{(2)}(x) = Z^{(1)}(x-1), \\
& \quad Z_1^{(1)}(x) = (0 \geq x \wedge Z^{(0)}(x)) \vee (x \geq 1 \wedge Z^{(2)}(x))\} \\
& \cup \{Z_2^{(i)}(x) = Z_{21}^{(i)}(x) \wedge X^{(i)}(x) \mid i \in \{0,1,2\}\} \\
& \cup \{Z_{21}^{(i)}(x) = x \geq 1 \mid i \in \{0,1,2\}\} \\
\mathcal{H}_1 = \ & (\{X^{(i)} \mid i \in \{0,1,2\}\}, \nu); \\
& (\{Y^{(i)}, Y_1^{(i)}, Y_2^{(i)}, Y_{21}^{(i)}, Z^{(i)}, Z_1^{(i)}, Z_2^{(i)}, Z_{21}^{(i)} \mid i \in \{0,1,2\}\}, \mu)
\end{aligned}$$

□

The following example has been taken from [15] (Figure 6, Bench 8), and modified to adjust the program syntax.

Example 6. Consider the program $\mathbf{P}_2 = (\mathbf{PC}_2, \{x\}, \mathbf{Code}_2)$, where $\mathbf{PC}_2 = \{0,1,2,3,4,5,6\}$ and $\mathbf{Code}_2$ consists of:

$$\begin{aligned}
& 0 \mapsto x := 1; \textbf{goto } 1, \\
& 1 \mapsto \textbf{if } * \textbf{ then goto } 1 \textbf{ else goto } 4, \\
& 2 \mapsto \textbf{if } * \textbf{ then goto } 5 \textbf{ else goto } 6, \\
& 3 \mapsto x := x; \textbf{goto } 3, \\
& 4 \mapsto x := 0; \textbf{goto } 2, \\
& 5 \mapsto x := 1; \textbf{goto } 3, \\
& 6 \mapsto x := 0; \textbf{goto } 3.
\end{aligned}$$

Consider the CTL* property $\mathtt{AG}(\mathtt{AFG}(x = 0) \vee \mathtt{AFG}(x = 1))$, which is expressed in the modal μ-calculus as:

$$\nu X.(\Box X \wedge ((\mu Y.\nu Z.\Box Y \vee (x = 0 \wedge \Box Z)) \vee (\mu U.\nu V.\Box U \vee (x = 1 \wedge \Box V)))).$$

The corresponding HMES is $\Xi_2 = (\mathbf{E}_2, \mathbf{H}_2)$ where:

$$\begin{aligned}
\mathbf{E}_2 = \{ \ & X = X_1 \wedge X_2, X_1 = \Box X, X_2 = Y \vee U, \\
& Y = Z, Z = Z_1 \vee Z_2, Z_1 = \Box Y, Z_2 = Z_{21} \wedge Z_{22}, Z_{21} = (x = 0), Z_{22} = \Box Z, \\
& U = V, V = V_1 \vee V_2, V_1 = \Box U, V_2 = V_{21} \wedge V_{22}, V_{21} = (x = 1), V_{22} = \Box V\} \\
\mathbf{H}_2 = \ & (\{X, X_1, X_2\}, \nu); (\{Y, U\}, \mu); (\{Z, Z_1, Z_2, Z_{21}, Z_{22}, V, V_1, V_2, V_{21}, V_{22}\}, \nu).
\end{aligned}$$

Our translation yields $\Phi_{\Xi_2,\mathbf{P}_2}$ shown in Figure 1. □

$$\Phi_{\Xi_2,\mathbf{P}_2} = (\mathcal{E}_2, \mathcal{H}_2)$$

$$\begin{aligned}
\mathcal{E}_2 = {} & \{X^{(i)}(x) = X_1^{(i)}(x) \wedge X_2^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{X_1^{(0)}(x) = X^{(1)}(1), X_1^{(1)}(x) = X^{(1)}(x) \wedge X^{(4)}(x), X_1^{(2)}(x) = X^{(5)}(x) \wedge X^{(6)}(x), \\
& \quad X_1^{(3)}(x) = X^{(3)}(x), X_1^{(4)}(x) = X^{(2)}(0), X_1^{(5)}(x) = X^{(3)}(1), X_1^{(6)}(x) = X^{(3)}(0)\} \\
& \cup \{X_2^{(i)}(x) = Y^{(i)}(x) \vee U^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{Y^{(i)}(x) = Z^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{Z^{(i)}(x) = Z_1^{(i)}(x) \vee Z_2^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{Z_1^{(0)}(x) = Y^{(1)}(1), Z_1^{(1)}(x) = Y^{(1)}(x) \wedge Y^{(4)}(x), Z_1^{(2)}(x) = Y^{(5)}(x) \wedge Y^{(6)}(x), \\
& \quad Z_1^{(3)}(x) = Y^{(3)}(x), Z_1^{(4)}(x) = Y^{(2)}(0), Z_1^{(5)}(x) = Y^{(3)}(1), Z_1^{(6)}(x) = Y^{(3)}(0)\} \\
& \cup \{Z_2^{(i)}(x) = Z_{21}^{(i)}(x) \wedge Z_{22}^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{Z_{21}^{(i)}(x) = x = 0 \mid i \in \mathbf{PC}_2\} \\
& \cup \{Z_{22}^{(0)}(x) = Z^{(1)}(1), Z_{22}^{(1)}(x) = Z^{(1)}(x) \wedge Z^{(4)}(x), Z_{22}^{(2)}(x) = Z^{(5)}(x) \wedge Z^{(6)}(x), \\
& \quad Z_{22}^{(3)}(x) = Z^{(3)}(x), Z_{22}^{(4)}(x) = Z^{(2)}(0), Z_{22}^{(5)}(x) = Z^{(3)}(1), Z_{22}^{(6)}(x) = Z^{(3)}(0)\} \\
& \cup \{U^{(i)}(x) = V^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{V^{(i)}(x) = V_1^{(i)}(x) \vee V_2^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{V_1^{(0)}(x) = U^{(1)}(1), V_1^{(1)}(x) = U^{(1)}(x) \wedge U^{(4)}(x), V_1^{(2)}(x) = U^{(5)}(x) \wedge U^{(6)}(x), \\
& \quad V_1^{(3)}(x) = U^{(3)}(x), V_1^{(4)}(x) = U^{(2)}(0), V_1^{(5)}(x) = U^{(3)}(1), V_1^{(6)}(x) = U^{(3)}(0)\} \\
& \cup \{V_2^{(i)}(x) = V_{21}^{(i)}(x) \wedge V_{22}^{(i)}(x) \mid i \in \mathbf{PC}_2\} \\
& \cup \{V_{21}^{(i)}(x) = x = 1 \mid i \in \mathbf{PC}_2\} \\
& \cup \{V_{22}^{(0)}(x) = V^{(1)}(1), V_{22}^{(1)}(x) = V^{(1)}(x) \wedge V^{(4)}(x), V_{22}^{(2)}(x) = V^{(5)}(x) \wedge V^{(6)}(x), \\
& \quad V_{22}^{(3)}(x) = V^{(3)}(x), V_{22}^{(4)}(x) = V^{(2)}(0), V_{22}^{(5)}(x) = V^{(3)}(1), V_{22}^{(6)}(x) = V^{(3)}(0)\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{H}_2 = {} & (\{X^{(i)}, X_1^{(i)}, X_2^{(i)} \mid i \in \mathbf{PC}_2\}, \nu); \\
& (\{Y^{(i)}, U^{(i)} \mid i \in \mathbf{PC}_2\}, \mu); \\
& (\{Z^{(i)}, Z_1^{(i)}, Z_2^{(i)}, Z_{21}^{(i)}, Z_{22}^{(i)}, V^{(i)}, V_1^{(i)}, V_2^{(i)}, V_{21}^{(i)}, V_{22}^{(i)} \mid i \in \mathbf{PC}_2\}, \nu).
\end{aligned}$$

Fig. 1. $\Phi_{\Xi_2,\mathbf{P}_2}$ in Example 6

$$\frac{n \in Z}{E[*] \xrightarrow{\epsilon}_D E[n]} \qquad \frac{(f(x_1,\ldots,x_k) = e) \in D}{E[f(n_1,\ldots,n_k)] \xrightarrow{\epsilon}_D E[[n_1/x_1,\ldots,n_k/x_k]e]}$$

$$\frac{a \notin Z \qquad \mathbf{val}(a) = n}{E[a] \xrightarrow{\epsilon}_D E[n]} \qquad \frac{}{E[\mathbf{let}\ x = n\ \mathbf{in}\ e] \xrightarrow{\epsilon}_D E[[n/x]e]}$$

$$\frac{}{E[A; e] \xrightarrow{A}_D E[e]} \qquad \frac{n \geq 0}{E[\mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2] \xrightarrow{\epsilon}_D E[e_1]}$$

$$\frac{n < 0}{E[\mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2] \xrightarrow{\epsilon}_D E[e_2]}$$

Fig. 2. Operational Semantics. E ranges over the set of evaluation contexts, defined by $E ::= [\,] \mid \mathbf{let}\ x = E\ \mathbf{in}\ e$.

3.2 Linear-Time Property Verification of Recursive Programs

Target Language and Verification Problem We first define a language of first-order recursive programs with non-determinism. The syntax of programs is given by:

$$\begin{aligned} &D(\text{programs}) ::= \{f_1(x_1,\ldots,x_{k_1}) = e_1, \ldots, f_\ell(x_1,\ldots,x_{k_\ell}) = e_\ell\} \\ &e(\text{expressions}) ::= a \mid * \mid A; e \mid f(v_1,\ldots,v_k) \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \\ &\qquad\qquad\qquad\qquad \mid \mathbf{if}\ v \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \\ &a ::= v \mid a_1\ \mathbf{op}\ a_2 \qquad v ::= n \mid x \end{aligned}$$

The expression $*$ evaluates to an integer in a non-deterministic manner. The expression $A; e$ raises an *event* A and evaluates e. Here, we assume a finite set of events; they are referred to by temporal property specifications (expressed by Büchi automata below). The other expressions are standard and should be self-explanatory. In a function definition $f_i(x_1,\ldots,x_{k_i}) = e_i$, variables $x_1,\ldots x_{k_i}$ and functions $f_1,\ldots,f_\ell$ are bound in e_i and we assume a given program is closed.

In a program $D = \{f_1(x_1,\ldots,x_{k_1}) = e_1,\ldots,f_\ell(x_1,\ldots,x_{k_\ell}) = e_\ell\}$, we assume that $\{f_1,\ldots,f_\ell\}$ contains the special function name **main** of the "main" function with arity 0, that **main**() never terminates, and every infinite reduction sequence generates an infinite sequence of events.[6]

Operational Semantics. The transition relation $e \xrightarrow{\xi}_D e'$ (where ξ is either A or ϵ) is defined in Figure 2. Here, $\mathbf{val}(a)$ denotes the value of an integer arithmetic expression a. We write $e \overset{w}{\Longrightarrow}_D e'$ if $w = \xi_1 \cdots \xi_\ell$ and $e_{i-1} \xrightarrow{\xi_i}_D e_i$ for each $i \in \{1,\ldots,\ell\}$, with $e = e_0$ and $e' = e_i$. Here, we treat ϵ as the empty word.

We write $\mathcal{L}(D)$ for the set of infinite sequences $A_1A_2A_3\cdots$ such that

$$\mathbf{main}() \overset{A_1}{\Longrightarrow}_D e_1 \overset{A_2}{\Longrightarrow}_D e_2 \overset{A_3}{\Longrightarrow}_D \cdots .$$

[6] The assumption on non-termination is guaranteed by renaming **main** to **main**′, and adding the function definitions $\mathbf{main}() = \mathbf{let}\ x = \mathbf{main}'()\ \mathbf{in}\ \mathit{loop}()$ and $\mathit{loop}() = A_{\mathit{call}}; \mathit{loop}$; the last assumption is guaranteed by restricting the righthand side of each function definition to an expression of the form $A_{\mathit{call}}; e$.

Example 7. Consider the program D_0 consisting of the following function definitions.

$$\mathbf{main}() = \mathbf{let}\ x = *\ \mathbf{in}\ \mathtt{f}(x)$$
$$\mathtt{f}(x) = \mathbf{let}\ r = \mathtt{g}\ x\ \mathbf{in}\ (\mathtt{A}; \mathtt{f}(r)) \qquad \mathtt{g}(x) = \mathtt{B}; \mathbf{if}\ x \geq 0\ \mathbf{then}\ \mathtt{g}\ (x-1)\ \mathbf{else}\ 5$$

It has, for example, the following reduction sequence:

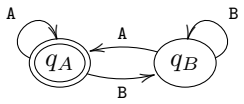
$$\mathbf{main}() \stackrel{\epsilon}{\Longrightarrow} \mathtt{f}(0) \stackrel{\epsilon}{\longrightarrow} \mathbf{let}\ r = \mathtt{g}(0)\ \mathbf{in}\ (\mathtt{A}; \mathtt{f}(r)) \stackrel{\mathtt{B}}{\longrightarrow} \mathbf{let}\ r = \mathtt{g}(-1)\ \mathbf{in}\ (\mathtt{A}; \mathtt{f}(r))$$
$$\stackrel{\epsilon}{\Longrightarrow} \mathtt{A}; \mathtt{f}(5) \stackrel{\mathtt{A}}{\longrightarrow} \mathtt{f}(5) \stackrel{\epsilon}{\longrightarrow} \cdots$$

The set $\mathcal{L}(D_0)$ of infinite event sequences generated by D_0 is $\{\mathtt{B}^n(\mathtt{AB}^6)^\omega \mid n \geq 1\}$.

Verification Problem We are interested in the verification of linear-time properties, expressible by using Büchi automata. Let us recall the definition of Büchi automata.

Definition 1 (Büchi automata). *A (non-deterministic)* Büchi automaton $\mathcal{A}$ *is a quintuple* $(\Sigma, Q, \Delta, q_0, F)$*, where (i)* Σ *is a set of input symbols, (ii)* Q *is a set of states, (iii)* $\Delta \in Q \times \Sigma \to 2^Q$ *is the transition function, (iv)* $q_0 \in Q$ *is the initial state, and (v)* $F \subseteq Q$ *is the set of final states. An* ω*-word* $w = A_1 A_2 \cdots \in \Sigma^\omega$ *is* accepted *by* $\mathcal{A}$ *if there exists an infinite sequence of states* $q'_0 q'_1 q'_2 \cdots \in Q^\omega$*, such that (i)* $q'_0 = q_0$*, (ii)* $q'_j \in \Delta(q'_{j-1}, A_j)$ *for each* $j \geq 1$*, and (iii)* $\forall j \in J. q'_j \in F$ *holds for an infinite subset* J *of natural numbers (in other words, final states are visited infinitely often). We write* $\mathcal{L}(\mathcal{A})$ *for the set of* ω*-words accepted by* $\mathcal{A}$*.*

Example 8. Let $\mathcal{A}_0 = (\{\mathtt{A}, \mathtt{B}\}, \{q_A, q_B\}, \Delta, q_A, \{q_A\})$ where $\Delta(q_A, \mathtt{A}) = \Delta(q_B, \mathtt{A}) = \{q_A\}$ and $\Delta(q_A, \mathtt{B}) = \Delta(q_B, \mathtt{B}) = \{q_B\}$. The automaton is depicted as follows.



$\mathcal{A}_0$ accepts an infinite word $w \in \{\mathtt{A}, \mathtt{B}\}^\omega$ just if w contains infinitely many $\mathtt{A}$'s.

We are interested in the following verification problem: Given a program D and a Büchi automaton $\mathcal{A}$, does $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$ hold? In a typical verification context, $\mathcal{L}(\mathcal{A})$ denotes the set of *invalid* infinite sequences of events, and the question $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \stackrel{?}{\neq} \emptyset$ asks whether D may generate an invalid infinite sequence. The goal of the rest of this section is to characterize the condition $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$ by a fixpoint formula $\varphi_{D,\mathcal{A}}$.

Overview of the Reduction Through an Example Consider the program D_0 and automaton $\mathcal{A}_0$ in Examples 7 and 8. Suppose we wish to verify that $\mathcal{L}(D_0) \cap \mathcal{L}(\mathcal{A}_0) \neq \emptyset$.

For each function $f \in \{\mathbf{main}, \mathtt{f}, \mathtt{g}\}$, we construct the following predicates:

– $f_{q,b,q'}(x, r)$ for each $q, q' \in \{q_A, q_B\}, b \in \mathbf{B}$. Intuitively, $f_{q,b,q'}(x, r)$ means that $f(x)$ may generate an event sequence that changes the state of the automaton from q to q', and returns r. The Boolean value b represents whether a final state is visited by the automaton during the state changes from q to q' (excluding the state q). For example, since $\mathtt{g}(-1) \xrightarrow{\mathtt{B}}_{D_0} 5$, $\mathtt{g}_{q,\mathtt{ff},q_B}(-1, 5)$ should hold for $q \in \{q_A, q_B\}$.

– $f_{q,b}(x)$ for each $q \in \{q_A, q_B\}, b \in \mathbf{B}$. Intuitively, $f_{q,b}(x)$ means that $f(x)$ may generate an event sequence that can be accepted by the automaton from the state q. The Boolean value b is determined by the calling context of f; it represents whether a final state has been visited since the parent recursive call of f. For example, $f_{q,b}(n)$ should hold for any integer n, $q \in \{q_A, q_B\}$, and $b \in \mathbf{B}$, as $f(n)$ generates an event sequence that contains infinitely many $\mathtt{A}$'s, which is accepted by $\mathcal{A}_0$. On the other hand, $\mathtt{g}_{q,b}(n)$ does not hold, as $\mathtt{g}(n)$ cannot generate an infinite sequence.

The predicates above can be systematically constructed from function definitions. For our running example, let us first construct $\mathtt{g}_{q,b,q'}$. Since $\mathtt{g}(x)$ generates only events $\mathtt{B}$ and returns r if either (i) $x \geq 0$ and $g(x-1)$ returns r, or (ii) $x < 0$ and $r = 5$. Thus, $\mathtt{g}_{q,b,q'}$ should satisfy:

$$\begin{aligned}
\mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) &= (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x-1, r)) \vee (x < 0 \wedge r = 5)\\
\mathtt{g}_{q_B,\mathtt{ff},q_B}(x, r) &= (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x-1, r)) \vee (x < 0 \wedge r = 5)\\
\mathtt{g}_{q,b,q'}(x, r) &= \mathtt{ff} \quad (\text{if } q' = q_A \text{ or } b = \mathtt{tt}).
\end{aligned}$$

Notice that the above equations are recursive. Since we are concerned about termination, $\mathtt{g}_{q,b,q'}$ is defined as the *least* solution of the equations above.

Using $\mathtt{g}_{q,b,q'}$ above, the equation for $\mathtt{f}_{q_A,\mathtt{tt}}$ is given as follows.

$$\mathtt{f}_{q_A,\mathtt{tt}}(x) = \exists r.(\mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) \wedge \mathtt{f}_{q_A,\mathtt{tt}}(r)).$$

This is because $\mathtt{f}(x)$ generates an infinite event sequence accepted from q_A by $\mathcal{A}_0$ if $\mathtt{g}(x)$ terminates and returns some r, and then $\mathtt{f}(r)$ generates an event sequence accepted from q_A. This time, $\mathtt{f}_{q_A,\mathtt{tt}}$ should be defined as the *greatest* solution for the above equation, since the automaton visits a final state (as indicated by the subscript $\mathtt{tt}$ for the predicate $\mathtt{f}_{q_A,\_}$) each time $\mathtt{f}$ is expanded. In general, $f_{q,b}$ is defined as the *greatest* solution if $b = \mathtt{tt}$, and as the *least* solution if $b = \mathtt{ff}$.

Based on the discussion above, $\Phi_{D_0,\mathcal{A}_0}$ is given as:

$$\begin{aligned}
&\{\mathbf{main}_{q_A,\mathtt{tt}}() =_\nu \exists x.\mathtt{f}_{q_A,\mathtt{ff}}(x),\\
&\quad \mathtt{f}_{q_A,\mathtt{tt}}(x) =_\nu \exists r.(\mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) \wedge \mathtt{f}_{q_A,\mathtt{tt}}(r))\};\\
&\{\mathtt{f}_{q_A,\mathtt{ff}}(x) =_\mu \exists r.(\mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) \wedge \mathtt{f}_{q_A,\mathtt{tt}}(r)),\\
&\quad \mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) =_\mu (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x-1, r)) \vee (x < 0 \wedge r = 5),\\
&\quad \mathtt{g}_{q_B,\mathtt{ff},q_B}(x, r) =_\mu (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x-1, r)) \vee (x < 0 \wedge r = 5)\}.
\end{aligned}$$

General Construction of $\Phi_{D,\mathcal{A}}$ We now formalize the general construction of the HES $\Phi_{D,\mathcal{A}}$. Below we fix a Büchi automaton $\mathcal{A} = (\Sigma, Q, \Delta, q_0, F)$.

As explained in the overview, for each function definition $f(\widetilde{x}) = e_f$, we construct predicates $f_{q,b,q'}(\widetilde{x}, r)$ and $f_{q,b}(\widetilde{x})$. To obtain equations for those predicates, we convert each subexpression e of e_f to formulas $[e]_{q,b,q',r}$ and $[e]_{q,b}$, where $q, q' \in Q$, $b \in \mathbf{B}$ and r is an integer variable. Intuitively, the formula $[e]_{q,b,q',r}$ means that there is a terminating execution sequence of e which generates a finite sequence of events that changes the state of $\mathcal{A}$ from q to q', and returns the integer r. The Boolean parameter b expresses whether an accepting state is visited during the automaton's transitions from q to q' (excluding the start state q). $[e]_{q,b}$ means that there is an infinite execution sequence of e that generates an infinite sequence of events accepted from q. The Boolean parameter b records information on whether an accepting state has been visited since the parent recursive call; the Boolean parameter is used to choose the Boolean parameter b' of $f_{q,b'}$.

The formula $[e]_{q,b,q',r}$ is defined by induction on the structure of e, as follows.

$$[a]_{q,b,q',r} = \begin{cases} a = r \text{ if } q = q' \text{ and } b = \mathtt{ff} \\ \mathtt{ff} \quad \text{otherwise} \end{cases} \qquad [*]_{q,b,q',r} = \begin{cases} \mathtt{tt} \text{ if } q = q' \text{ and } b = \mathtt{ff} \\ \mathtt{ff} \text{ otherwise} \end{cases}$$
$$[A; e]_{q,b,q',r} = \bigvee\{[e]_{q'',b',q',r} \mid q'' \in \Delta(q, A), b' \in \mathbf{B}, b' \vee (q'' \in F) = b\}$$
$$[f(a_1, \ldots, a_k)]_{q,b,q',r} = f_{q,b,q'}(a_1, \ldots, a_k, r)$$
$$[\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2]_{q,b,q',r} =$$
$$\quad \bigvee\{\exists x.([e_1]_{q,b_1,q'',x} \wedge [e_2]_{q'',b_2,q',r}) \mid q'' \in Q, b_1, b_2 \in \mathbf{B}, b = b_1 \vee b_2\}$$
$$[\mathbf{if}\ a \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]_{q,b,q',r} = (a \geq 0 \wedge [e_1]_{q,b,q',r}) \vee (a < 0 \wedge [e_2]_{q,b,q',r})$$

We explain a few cases. Since a immediately evaluates to an integer, $[a]_{q,b,q',r}$ is true just if $a = r$, $q = q'$, and $b = \mathtt{ff}$. In the translation of $A; e$, q'' is the state of $\mathcal{A}$ after the event A has occurred. The case for a function call $f(a_1, \ldots, a_k)$ is based on the intuition on the predicate $f_{q,b,q'}$ explained in Section 3.2. The translation for $e \equiv \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$ is based on the intuition that e evaluates to r just if e_1 evaluates to some integer x, and then e_2 evaluates to r; q'' is the intermediate state of $\mathcal{A}$ when e_1 has been evaluated.

The formula $[e]_{q,b}$ is also inductively defined as follows.

$$[a]_{q,b} = \mathtt{ff} \qquad [*]_{q,b} = \mathtt{ff} \qquad [A; e]_{q,b} = \bigvee\{[e]_{q',b\vee(q'\in F)} \mid q' \in \Delta(q, A)\}$$
$$[f(a_1, \ldots, a_k)]_{q,b} = f_{q,b}(a_1, \ldots, a_k)$$
$$[\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2]_{q,b} = [e_1]_{q,b} \vee (\bigvee\{\exists x.[e_1]_{q,b',q',x} \wedge [e_2]_{q',b\vee b'} \mid q' \in Q, b' \in \mathbf{B}\})$$
$$[\mathbf{if}\ a \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]_{q,b} = (a \geq 0 \wedge [e_1]_{q,b}) \vee (a < 0 \wedge [e_2]_{q,b})$$

When $e = a$ or $*$, $[e]_{q,b} = \mathtt{ff}$ since e does not generate an infinite event sequence. In the translation of $A; e$, we update the state and accumulate (by $b \vee (q' \in F)$) information on whether an accepting state has been visited. For a function call $f(a_1, \ldots, a_k)$, the Boolean parameter b is used to annotate f, so that $f_{q,b}$ is defined as the greatest fixpoint if $b = \mathtt{tt}$ (which means that an accepting state has been visited since the last recursive call), and otherwise defined as the least fixpoint. The translation for $e \equiv \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$ is based on the intuition that e diverges either if e_1 diverges, or if e_1 evaluates to an integer x and e_2 diverges.

Using $[e]_{q,b,q',r}$ and $[e]_{q,b}$, we define $\mathcal{E}_{D,b}, \mathcal{P}_{D,\mathtt{fin}}, \mathcal{E}_{D,\mathtt{fin}}$ and $\mathcal{P}_{D,b}$ ($b \in \mathbf{B}$) by:

$$\begin{aligned}
\mathcal{E}_{D,b} &= \{f_{q,b}(\widetilde{x}) = [e]_{q,\mathtt{ff}} \mid (f(\widetilde{x}) = e) \in D, q \in Q\} \\
\mathcal{P}_{D,b} &= \{f_{q,b} \mid (f(\widetilde{x}) = e) \in D, q \in Q\} \\
\mathcal{E}_{D,\mathtt{fin}} &= \{f_{q,b,q'}(\widetilde{x}, r) = [e]_{q,b,q',r} \mid (f(\widetilde{x}) = e) \in D, q, q' \in Q, b \in \mathbf{B}\} \\
\mathcal{P}_{D,\mathtt{fin}} &= \{f_{q,b,q'} \mid (f(\widetilde{x}) = e) \in D, q, q' \in Q, b \in \mathbf{B}\}.
\end{aligned}$$

Finally, we define $\Phi_{D,\mathcal{A}} = (\mathcal{E}_D, \mathcal{H}_D)$ by:

$$\mathcal{E}_D = \mathcal{E}_{D,\mathtt{tt}} \cup \mathcal{E}_{D,\mathtt{ff}} \cup \mathcal{E}_{D,\mathtt{fin}} \qquad \mathcal{H}_D = (\mathcal{P}_{D,\mathtt{tt}}, \nu); (\mathcal{P}_{D,\mathtt{ff}} \cup \mathcal{P}_{D,\mathtt{fin}}, \mu).$$

As indicated above, the alternation depth (between ν and μ) of $\Phi_{D,\mathcal{A}}$ is 2.

The following theorem states the correctness of the construction of $\Phi_{D,\mathcal{A}}$. A proof is given in Appendix A.

Theorem 1. *Let D be a program and $\mathcal{A}$ be a Büchi automaton. Then $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if $\Phi_D \models \mathbf{main}_{q_0,\mathtt{tt}}()$.*

Example 9. We have already given an example of the construction of $\Phi_{D,\mathcal{A}}$ in Section 3.2. We give another simple example here, which may help the reader understand the role of the Boolean parameter b in $f_{q,b}$. Recall the automaton $\mathcal{A}_0$ in Example 8, and consider the program D_1 that consists of the single function definition $\mathbf{main}() = \mathtt{A}; \mathbf{main}()$. Then, $\Phi_{D_1,\mathcal{A}_0}$ is:

$$\{\mathbf{main}_{q_A,\mathtt{tt}}() =_\nu \mathbf{main}_{q_A,\mathtt{tt}}()\}; \{\mathbf{main}_{q_A,\mathtt{ff}}() =_\mu \mathbf{main}_{q_A,\mathtt{tt}}()\}.$$

(We omit the equations for $\mathbf{main}_{q_B,b}$.) Thus, $\Phi_{D_1,\mathcal{A}_0} \models \mathbf{main}_{q_A,\mathtt{tt}}()$. Indeed, D_1 generates $\mathtt{A}^\omega$, which is accepted by $\mathcal{A}_0$. The reason why $\mathbf{main}_{q_A,\_}$ in the bodies of the equations is annotated with $\mathtt{tt}$ is that an event $\mathtt{A}$ occurs (so, the automaton visits the accepting state q_A) before **main** is called in the body of the function definition.

In contrast, the program D_2 consisting of $\mathbf{main}() = \mathtt{B}; \mathbf{main}()$ is translated to: $\{\mathbf{main}_{q_A,\mathtt{tt}}() =_\nu \mathbf{main}_{q_B,\mathtt{ff}}()\}; \{\mathbf{main}_{q_B,\mathtt{ff}}() =_\mu \mathbf{main}_{q_B,\mathtt{ff}}()\}$. Thus, $\Phi_{D_2,\mathcal{A}_0} \not\models \mathbf{main}_{q_A,\mathtt{tt}}()$. Indeed, D_2 only generates $\mathtt{B}^\omega$, which is not accepted by $\mathcal{A}_0$. Note that $\mathbf{main}_{q_B,\_}$ in the bodies of the equations is annotated with $\mathtt{ff}$ because each call of $\mathbf{main}()$ is only preceded by an event $\mathtt{B}$; so, the automaton does not visit q_A. □

4 Proving Fixpoint Formulas by Reduction to CHC Solving

In this section, we describe our Mu2CHC approach to validity checking of fixpoint formulas. The method is based on a reduction to CHC solving [5]. As mentioned in Section 1, a main advantage of the approach is that we can reuse off-the-shelf CHC solvers such as Spacer [27] and HoIce [13].

Suppose that we wish to prove $\Phi \models \mathbf{main}()$. Without loss of generality, we can assume that the predicate **main** is bound by ν in Φ (otherwise, just introduce

a fresh predicate $\mathbf{main}'$, and prove $\{\mathbf{main}'() =_\nu \mathbf{main}()\}; \Phi \models \mathbf{main}())$. We can also assume that Φ contains no existential quantifiers, as existential quantifiers can be encoded by using μ; recall Remark 1. Below we present a method to transform Φ to another HES Φ' so that $\Phi' \models \mathbf{main}()$ implies $\Phi \models \mathbf{main}()$, and Φ' contains neither μ nor existential quantifiers. By the observation in Section 2.3, $\Phi' \models \mathbf{main}()$ can be reduced to CHC solving. Using a CHC solver as a backend, we obtain a sound procedure for proving $\Phi \models \mathbf{main}()$. To disprove $\Phi \models \mathbf{main}()$, it suffices to prove the dual problem $\overline{\Phi} \models \overline{\mathbf{main}}()$ (where $\overline{\mathbf{main}}$ is the predicate symbol that denotes the negation of $\mathbf{main}$ in $\overline{\Phi}$); thus, by running the sound procedure for proving $\Phi \models \mathbf{main}()$ and $\overline{\Phi} \models \overline{\mathbf{main}}()$ in parallel, we obtain a sound (but incomplete) decision procedure.

Inspired by methods for proving termination by reduction to safety properties [34], we approximate μ-formulas (which can be considered generalization of the termination property) by ν-formulas (which can be considered generalization of safety properties). In particular, we pick the recent termination verification method of Fedyukovich et al. [21] and generalize it for our context.

Let us first consider a special case, where an HES consists of a single equation: $P(\widetilde{x}) =_\mu \varphi$. Recall that the semantics of P is the least fixpoint of $F = \lambda f \in \mathbf{Z}^k \to \mathbf{B}.\lambda \widetilde{v} \in \mathbf{Z}^k.[\![\varphi]\!]_{\{P \mapsto f, \widetilde{x} \to \widetilde{v}\}}$, where $k = \mathtt{ar}(P)$. Thus, the semantics of P can be under-approximated by $F^y(\lambda \widetilde{v}.\mathtt{ff})$ for any $y \geq 0$, and a greater value of y gives a better approximation. With this in mind, we prepare a new predicate P' and construct a new equation:

$$P'(y, \widetilde{x}) =_\nu y > 0 \wedge \varphi',$$

where φ' is the formula obtained from φ by replacing each formula of the form $P(\widetilde{t})$ with $P'(y-1, \widetilde{t})$. The predicate $\lambda \widetilde{x}.P'(y, \widetilde{x})$ corresponds to $F^y(\lambda \widetilde{v}.\mathtt{ff})$ above (in fact, one can prove that the semantics of $\lambda \widetilde{x}.P'(y, \widetilde{x})$ is equivalent to $F^y(\lambda \widetilde{v}.\mathtt{ff})$ by induction on y), and thus $P'(y, \widetilde{x}) \Rightarrow P(\widetilde{x})$ for any $y \in \mathbf{Z}$. To prove a formula of the form $C[P(\widetilde{a})]$ (here, C is a formula with a hole, and we write $C[\varphi]$ for the formula obtained by filling the hole with φ), it suffices to prove $C[\forall y.(y \geq a'_1 \wedge \cdots \wedge y \geq a'_k \Rightarrow P'(y, \widetilde{a}))]$, where $a'_1, \ldots, a'_k$ are arbitrary arithmetic expressions constructed by using variables available in the hole of C. Note that $\forall y.(y \geq a'_1 \wedge \cdots \wedge y \geq a'_k \Rightarrow P'(y, \widetilde{a}))$, which is equivalent to $P'(\max(a'_1, \ldots, a'_k), \widetilde{a})$, implies $P(x)$, and that (the semantics of) C is monotonic with respect to the hole position, since there is no connective for negation. Thus, we have reduced the validity checking problem for a least fixpoint formula with that of a greatest fixpoint formula in a sound (but incomplete) manner.

Remark 2. In $\forall y.(y \geq a'_1 \wedge \cdots \wedge y \geq a'_k \Rightarrow P'(y, \widetilde{a}))$, the bounds $a'_1, \ldots, a'_k$ can be chosen heuristically. A nice point about using multiple bounds is that we can monotonically increase the precision of approximation, by adding new elements to the set $\{a'_1, \ldots, a'_k\}$. This advantage is analogous to that of disjunctive well-founded relations over well-founded relations in the context of termination verification [34].

Example 10. Consider:

$$P(x) =_\mu x = 0 \vee P(x-1)$$

and suppose that we wish to prove $\forall z.z < 0 \vee P(z)$. We define a new predicate P' by:

$$P'(y, x) =_\nu y > 0 \wedge (x = 0 \vee P'(y - 1, x - 1)),$$

and change the goal to $\forall z.z < 0 \vee (\forall y.y \geq z + 1 \Rightarrow P'(y, z))$. One can reduce its validity to the satisfiability of the following CHC:

$$\overline{P'}(y, x) \Leftarrow y \leq 0 \vee (x \neq 0 \wedge \overline{P'}(y - 1, x - 1))$$
$$\mathtt{ff} \Leftarrow z \geq 0 \wedge y \geq z + 1 \wedge \overline{P'}(y, z).$$

It is satisfiable with $\overline{P'}(y, x) \equiv y \leq 0 \vee x < 0 \vee y < x + 1$; hence we know the original formula $\forall z.z < 0 \vee P(z)$ is valid. □

In the general case, we replace each layer of μ-equations with ν-equations one by one. Assume that a given HES Φ is

$$\Phi_1; \{P_1(\widetilde{x}_1) =_\mu \varphi_1, \ldots, P_k(\widetilde{x}_k) =_\mu \varphi_k\};$$
$$\{P_{k+1}(\widetilde{x}_{k+1}) =_\nu \varphi_{k+1}, \ldots, P_{k+\ell}(\widetilde{x}_{k+\ell}) =_\nu \varphi_{k+\ell}\}.$$

As in the special case, we approximate the least fixpoint $\mathbf{FP}_\mu(F)$ for the values of $P_1, \ldots, P_k$ with a finite approximation $F^y(\bot)$. To this end, we replace Φ with the following HES Φ':

$$\Phi'_1; \{P'_1(y, \widetilde{x}_1) =_\mu y > 0 \wedge \varphi'_1, \ldots, P'_k(y, \widetilde{x}_k) =_\mu y > 0 \wedge \varphi'_k\};$$
$$\{P'_{k+1}(y, \widetilde{x}_{k+1}) =_\nu \varphi'_{k+1}, \ldots, P'_{k+\ell}(y, \widetilde{x}_{k+\ell}) =_\nu \varphi'_{k+\ell}\}.$$

Here,

– φ'_i $(1 \leq i \leq k + \ell)$ is the formula obtained from φ_i by replacing each subformula of the form $P_j(\widetilde{a})$ with $P'_j(y - 1, \widetilde{a})$ if $1 \leq j \leq k$, and with $P'_j(y, \widetilde{a})$ if $k+1 \leq j \leq k+\ell$). Intuitively, $P'_i(y, \_)$ $1 \leq i \leq k$ is the y-th approximation $F^y(\bot)$ of the least fixpoint of F; hence y is decremented each time P_i is recursively called. For $P'_i(y, \widetilde{x}_i)$ for $k + 1 \leq i \leq k + \ell$ approximates $P_i(\widetilde{x}_i)$ by approximating $P_j(\widetilde{x}_j)$ with $P'_j(y, \widetilde{x}_j)$ for $1 \leq j \leq k$ (recall that the semantics of $P_{k+1}, \ldots, P_{k+\ell}$ are parameterized by those of $P_1, \ldots, P_k$).

– Φ'_1 is the HES obtained from Φ_1 by replacing each formula of the form $P_i(\widetilde{a})$ $(1 \leq i \leq k + \ell)$ with $\forall y.y \geq a'_1 \wedge \cdots \wedge y \geq a'_m \Rightarrow P'_i(y, \widetilde{a})$, where $a'_1, \ldots, a'_m$ are expressions consisting of the variables available at the position of $P_i(\widetilde{a})$.

By the construction above, $P'_i(y, \widetilde{x}_i) \Rightarrow P_i(\widetilde{x}_i)$ holds for every $y, \widetilde{x}_1 \in \mathbf{Z}$; hence Φ' is an under-approximation of Φ. By repeatedly applying the transformation above to Φ, we get an HES Φ' such that Φ' contains neither μ nor $\exists$, and $[\![\Phi']\!] \sqsubseteq [\![\Phi]\!]$.

Example 11. Recall the HES Φ_1 in Section 3.2 (with some simplification):

$$\{\mathbf{main}_{q_A,\mathtt{tt}}() =_\nu \exists x.\mathtt{f}_{q_A,\mathtt{tt}}(x), \quad \mathtt{f}_{q_A,\mathtt{tt}}(x) =_\nu \exists r.(\mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) \wedge \mathtt{f}_{q_A,\mathtt{tt}}(r))\};$$
$$\{\mathtt{g}_{q_A,\mathtt{ff},q_B}(x, r) =_\mu (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x - 1, r)) \vee (x < 0 \wedge r = 5),$$
$$\mathtt{g}_{q_B,\mathtt{ff},q_B}(x, r) =_\mu (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x - 1, r)) \vee (x < 0 \wedge r = 5)\}.$$

By encoding $\exists$ with μ, we obtain:

$$\begin{array}{l}\{\mathbf{main}_{q_A,\mathtt{tt}}() =_\nu P(0), \quad \mathtt{f}_{q_A,\mathtt{tt}}(x) =_\nu Q(0,x)\};\\ \{P(x) =_\mu \mathtt{f}_{q_A,\mathtt{tt}}(x) \vee P(x+1) \vee P(x-1),\\ \quad Q(r,x) =_\mu (\mathtt{g}_{q_A,\mathtt{ff},q_B}(x,r) \wedge \mathtt{f}_{q_A,\mathtt{tt}}(r)) \vee Q(r+1,x) \vee Q(r-1,x),\\ \quad \mathtt{g}_{q_A,\mathtt{ff},q_B}(x,r) =_\mu (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x-1,r)) \vee (x < 0 \wedge r = 5),\\ \quad \mathtt{g}_{q_B,\mathtt{ff},q_B}(x,r) =_\mu (x \geq 0 \wedge \mathtt{g}_{q_B,\mathtt{ff},q_B}(x-1,r)) \vee (x < 0 \wedge r = 5)\}.\end{array}$$

By the transformation above, we obtain Φ_1':

$$\begin{array}{l}\{\mathbf{main}_{q_A,\mathtt{tt}}() =_\nu \forall y.(y \geq 1 \Rightarrow P'(y,0)),\\ \quad \mathtt{f}_{q_A,\mathtt{tt}}(x) =_\nu \forall y.(y \geq x+6 \Rightarrow Q'(y,0,x)),\\ \quad P'(y,x) =_\nu y > 0 \wedge (\mathtt{f}_{q_A,\mathtt{tt}}(x) \vee P'(y-1,x+1) \vee P'(y-1,x-1)),\\ \quad Q'(y,r,x) =_\nu y > 0 \wedge ((\mathtt{g}_{q_A,\mathtt{ff},q_B}(y-1,x,r) \wedge \mathtt{f}_{q_A,\mathtt{tt}}(y-1,r))\\ \qquad\qquad \vee Q'(y-1,r+1,x) \vee Q'(y-1,r-1,x)),\\ \quad \mathtt{g}'_{q_A,\mathtt{ff},q_B}(y,x,r) =_\nu y > 0\\ \qquad\qquad \wedge((x \geq 0 \wedge \mathtt{g}'_{q_B,\mathtt{ff},q_B}(y-1,x-1,r)) \vee (x < 0 \wedge r = 5)),\\ \quad \mathtt{g}'_{q_B,\mathtt{ff},q_B}(y,x,r) =_\nu y > 0\\ \qquad\qquad \wedge((x \geq 0 \wedge \mathtt{g}'_{q_B,\mathtt{ff},q_B}(y-1,x-1,r)) \vee (x < 0 \wedge r = 5))\}.\end{array}$$

Since $\Phi_1' \models \mathbf{main}_{q_A,\mathtt{tt}}()$, we know $\Phi_1 \models \mathbf{main}_{q_A,\mathtt{tt}}()$. □

Remark 3. As explained above, the idea of our translation is to approximate the least fixpoint $\mathbf{FP}_\mu(F)$ with $F^k(\bot)$. This is too conservative, when (i) the least fixpoint is not reached in the ω-step (i.e., when $\mathbf{FP}_\mu(F) \neq F^\omega(\bot)$), or (ii) the bound k is too large to express it and for the underlying CHC solver to reason about (e.g. when k is expressed by the Ackermann function). One way to overcome this problem is to represent a bound as a tuple of integers. For example, $P(\widetilde{x}) =_\mu \varphi$ can be approximated by $P'(y_1, y_2, \widetilde{x})$, which is defined by:

$$P'(y_1, y_2, \widetilde{x}) =_\nu y_1 > 0 \wedge y_2 > 0 \wedge \varphi',$$

where φ' is the formula obtained from φ' by replacing each subformula of the form $P(\widetilde{a})$ with

$$P'(y_1, y_2 - 1, \widetilde{a}) \vee \forall y_2'.(y_2' \geq \max(a_1', \ldots, a_k') \Rightarrow P'(y_1 - 1, y_2', \widetilde{a})).$$

Note that when the value of y_1 is decreased, the value of y_2 can be reset. This corresponds to the use of a lexicographic ranking function in termination verification. □

Figure 3 shows pseudo code of our overall procedure. The procedure `CheckValidity` takes as input an HES Φ and an entry predicate **main** and returns whether $\Phi \models \mathbf{main}()$ holds. If Φ is ν-only (i.e., it contains neither $\exists$ nor μ), then the procedure converts the problem to the corresponding CHC satisfiability problem, and calls a backend CHC solver. Similarly, if Φ is μ-only (i.e., it contains neither $\forall$ nor ν), then the procedure makes the de Morgan dual of the problem by `MakeDual`, converts it to CHC, and calls a CHC solver; in this case,

```
CheckValidity(Φ, main){ /* Φ: HES, main: Entry formula */
  if Φ is a ν-only HES then return CHCsolver(toCHC(MakeDual(Φ,main)))
  else if Φ is a μ-only HES then return not(CHCsolver(toCHC(Φ,main)))
  else return (CheckSub(Φ, main) || not(CheckSub(MakeDual(Φ,main))));
}
CheckSub(Φ, main){
  (Φ',main') := ElimMu(Φ,main)
  while(true) {
    if CHCsolver(toCHC(Φ',main')) then return true
    else (Φ',main') := IncreaseBounds(Φ',main'); }
}
```

Fig. 3. Mu2CHC Procedure

the final result is the negation of the output of the CHC solver. The remaining is the case where Φ has alternations between μ and ν. In this case, the procedure runs the subprocedure `CheckSub` for proving the original problem and its dual in parallel. As described above, `CheckSub` approximates a given HES Φ to a ν-only HES Φ' and then converts Φ' to CHC. Due to the under-approximation, the result is valid only if the CHC solver returns true (which means the formula is valid); if the CHC solver returns false or time-outs, the procedure increases bounds ($a'_1, \ldots, a'_k$ used for eliminating μ) and repeats the loop.

5 Implementation and Evaluation

We have implemented a validity checking tool MU2CHC for the fixpoint logic in OCaml, based on the method in Section 4. We use Spacer [27] and HoIce [13] as the backend CHC solvers of MU2CHC. In addition, we have also implemented a translator from CTL verification problems for C programs to Mu-Arithmetic formulas, which supports only a very small subset of C, just large enough to cover the benchmark programs of [17]. We have not yet implemented the translations described in Sections 3.1 and 3.2 (implementing them for a full-scale language is not difficult but tedious); thus, the outputs of those translations used in the experiments below have been obtained by hand.

As the set of bounds $\{a'_1, \ldots, a'_k\}$ used for approximating μ-formulas by ν-formulas (recall Remark 2), MU2CHC uses $\{c_1x_1+...+c_nx_n+B \mid c_i \in \{-A, A\}\}$ where A, B are positive integers, and $x_1, \ldots, x_n$ are the variables in scope. Those bounds are equivalent to the single bound $A(|x_1|+...+|x_n|)+B$. MU2CHC first sets $A = 1, B = 10$, and doubles them each time the `IncreaseBounds` procedure in Figure 3 is called. In the implementation, an existentially-quantified formula $\exists x.\varphi$ is directly approximated by the formula $\forall x.x \geq a'_1 \wedge \cdots \wedge x \geq a'_k \Rightarrow P(x)$, where $P(x) =_\nu x \geq 0 \wedge (\varphi \vee [-x/x]\varphi \vee P(x-1))$ (rather than encoding it using μ as in Remark 1 and then approximating μ by ν). Note that $\forall x.x \geq a'_1 \wedge \cdots \wedge x \geq a'_k \Rightarrow P(x)$ is equivalent to $\exists x.x \leq \max(a'_1, \ldots, a'_k) \wedge \varphi$.

We have tested MU2CHC against our own benchmark set, and the standard benchmark set for CTL verification [17]. The tool was run on an Intel Core i5 2.7 GHz dual-core processor with main memory of 8 GB.

The first table in Table 1 shows the results on our own benchmark set. The columns Exp. and Act. show expected and actual results, respectively, where ✓, ✗, and ? denote "valid", "invalid", and "unknown", respectively. All the problems except ex4 contain nested ν and μ. The problems 1–6 encode some properties of integer arithmetic in the fixpoint logic. The problems 7–22 encode linear-time properties of recursive programs, based on the translation in Section 3 (with some hand-optimizations). In particular, 7 and 8 are $\Phi_{D_0,\mathcal{A}_0}$ in Section 3.2 and a variation of it. The problems 9-14 are from [23, 28, 31]. For those problems, the formulas encode the property "there is an (infinite) error trace that violates an expected linear-time property." Those formulas are *invalid*, since the original programs have actually no error trace. The rest of the problems (23–28) encode temporal property of while-programs, based on the translation in Section 3.1. Among them, the problems 23 and 24 verify the properties $\nu X.(x+y \geq 0 \wedge \Diamond\Diamond X)$ and $\nu X.\mu Y.\Box Y \vee (0 \geq x \wedge (\mu Z.\Diamond Z \vee (x \geq 1 \wedge X)))$ respectively, which we believe cannot be expressed in CTL*. The problem ex6 has been taken from a test case[7] of T2 [15], and verifies the CTL* property $\mathtt{AG}(\mathtt{AFG}(x = 0) \vee \mathtt{AFG}(x = 1))$. See Appendix B for more details on the benchmark set. Our tool could successfully check the validity of Mu-Arithmetic formulas, except the problem 22. It requires a reasoning about the divisibility predicate, which is not well handled by the underlying CHC solvers.

The table below in Table 1 shows the result for the "industrial" benchmark set from [17]; the result for the "small" benchmark set is provided in Appendix B. For comparison, we take the results from [17] verbatim (note that the execution time is measured by using a different processor).[8] As the table shows, our tool could successfully solve all the problems and outperforms [17] in most cases (note however the difference in the experimental environments). This may be a bit surprising, as our tool is not customized for CTL verification.

6 Related Work

As already mentioned, our work has been motivated by recent proposals of reductions from program verification to validity/satisfiability checking in fixpoint

[7] https://github.com/hkhlaaf/T2/blob/master/test/ctlstar_test.t2.

[8] There are some discrepancies on the verification results among [17], [3] and ours. We are not sure about this, but it is most likely because the benchmark set has accidentally been modified when it was passed around. We have taken the industrial set from that of E-HSF [3] provided by Andrey Rybalchenko. Note, however, that we found some discrepancies between the C programs and their encodings in the E-HSF; that explains the difference between the outputs of our tool and those of E-HSF [3]. The "small" set was provided by Eric Koskinen. For 26–28, the results are "invalid" for both φ and $\neg\varphi$ (as in [3]); this is not a contradiction, as the checked properties are of the form "*for all the initial states*, φ (or $\neg\varphi$) holds."

Table 1. Experimental Results. The upper table shows the results for our own benchmark set, and the lower table shows the results for the Industrial Set from [17].

Benchmark Name	Exp.	Act.	Time[s]	Benchmark Name	Exp.	Act.	Time[s]
1. simple-nest	✓	✓	0.21	15. infinite1	✓	✓	1.58
2. simple-nest-inv	✓	✓	0.19	16. infinite1b	✓	✓	2.97
3. lines1	✓	✓	1.72	17. infinite1c-invalid	✗	✗	1.64
4. lines2-invalid	✗	✗	0.27	18. infinite2	✓	✓	0.36
5. lines3	✓	✓	1.95	19. infinite3	✓	✓	0.13
6. lines4	✓	✓	1.66	20. intfun1-invalid	✗	✗	0.06
7. ex3	✓	✓	9.36	21. intfun2-invalid	✗	✗	0.06
8. ex3-forall	✓	✓	22.87	22. intfun3-invalid	✗	?	-
9. hofmann1	✗	✗	0.17	23. ex4	✓	✓	0.09
10. hofmann2	✗	✗	0.48	24. ex5	✓	✓	0.13
11. koskinen1_fo	✗	✗	0.27	25. ex6	✗	✗	1.58
12. koskinen2	✗	✗	1.50	26. ctl1	✓	✓	0.79
13. koskinen3	✗	✗	0.46	27. ctl2	✓	✓	5.50
14. intro	✗	✗	1.96	28. ctl2b-invalid	✗	✗	1.72

Problem ID and Property φ	$\models \varphi$					$\models \neg\varphi$				
	Exp.	[17]		MU2CHC		Exp.	[17]		MU2CHC	
		Act.	Time[s]	Act.	Time[s]		Act.	Time[s]	Act.	Time[s]
1. $\mathtt{AG}(p \Rightarrow \mathtt{AF}q)$	✓	✓	4.6	✓	0.40	✗	✗	12.5	✗	0.41
2. $\mathtt{AG}(p \Rightarrow \mathtt{AF}q)$	✗	✗	9.1	✗	0.10	✓	✓	3.5	✓	0.32
3. $\mathtt{AG}(p \Rightarrow \mathtt{EF}q)$	✓	✓	9.5	✓	0.23	✗	✗	18.1	✗	1.57
4. $\mathtt{AG}(p \Rightarrow \mathtt{EF}q)$	✗	✗	1.5	✗	0.65	✓	✓	105.7	✓	0.82
5. $\mathtt{AG}(p \Rightarrow \mathtt{AF}q)$	✓	✓	2.1	✓	0.49	✗	✗	6.5	✗	3.91
6. $\mathtt{AG}(p \Rightarrow \mathtt{AF}q)$	✗	✗	1.8	✗	0.15	✓	✓	1.2	✓	2.91
7. $\mathtt{AG}(p \Rightarrow \mathtt{EF}q)$	✓	✓	3.7	✓	4.91	✗	✗	8.7	✗	6.33
8. $\mathtt{AG}(p \Rightarrow \mathtt{EF}q)$	✗	✗	1.5	✗	5.55	✓	✓	5.6	✓	4.25
9. $\mathtt{AG}(p \Rightarrow \mathtt{AF}q)$	✓	✓	38.9	✓	0.65	✗	✗	1930.9	✗	3.27
10. $\mathtt{AG}(p \Rightarrow \mathtt{AF}q)$	✗	✗	148.0	✗	28.20	✓	✓	1680.7	✓	29.53
11. $\mathtt{AG}(p \Rightarrow \mathtt{EF}q)$	✓	✓	90.0	✓	0.42	✗	?	-	✗	2.69
12. $\mathtt{AG}(p \Rightarrow \mathtt{EF}q)$	✓	✗	107.8	✓	0.52	✗	?	-	✗	2.92
13. $\mathtt{AF}q \vee \mathtt{AF}p$	✗	✓	34.3	✗	0.16	✓	✗	62.3	✗	14.62
14. $\mathtt{AF}q \vee \mathtt{AF}p$	✗	✗	18.8	✗	0.20	✓	✓	7.6	✓	1.91
15. $\mathtt{EF}q \wedge \mathtt{EF}p$	✓	✓	1261.0	✓	21.87	✗	✗	0.9	✗	0.14
16. $\mathtt{EF}q \wedge \mathtt{EF}p$	✗	?	-	✗	1.80	✓	✓	0.6	✓	0.16
17. $\mathtt{AGAF}p$	✓	✓	596.7	✓	0.58	✗	✗	1471.7	✗	2.39
18. $\mathtt{AGAF}p$	✗	✗	65.1	✗	0.07	✓	✓	351.1	✓	0.23
19. $\mathtt{AGEF}p$	✗	?	-	✗	0.46	✓	✗	85.5	✓	0.38
20. $\mathtt{AGEF}p$	✗	?	-	✗	0.89	✓	✓	255.8	✓	0.52
21. $\mathtt{AGAF}p$	✗	?	-	✗	1.22	✓	✗	45.3	✓	0.29
22. $\mathtt{AGAF}p$	✗	✗	38.1	✗	0.13	✓	✓	35.2	✓	0.32
23. $\mathtt{AGEF}p$	✗	?	-	✗	0.11	✓	?	-	✓	0.11
24. $\mathtt{AGEF}p$	✗	✗	42.7	✗	0.11	✓	✓	30.2	✓	1.62
25. $p \Rightarrow \mathtt{AF}q$	✓	✓	70.2	✓	17.17	✗	✗	0.4	✗	0.13
26. $p \Rightarrow \mathtt{AF}q$	✗	✗	32.4	✗	0.84	✗	✓	4.5	✗	0.09
27. $p \Rightarrow \mathtt{EF}q$	✗	✓	18.5	✗	0.94	✗	✗	0.5	✗	0.09
28. $p \Rightarrow \mathtt{EF}q$	✗	✗	1.3	✗	0.07	✗	✓	0.3	✗	0.11

logics [3, 5, 7–9, 22, 26, 32, 40]. The idea of using CHC in program analysis or verification can actually be further traced back to earlier studies on constraint logic programming [19, 24, 33].

The combination of our method for proving Mu-Arithmetic formulas with the translation given in Section 3.1 yields an automated verification method for the full modal μ-calculus model checking of while-programs with infinite data. In contrast, the previous temporal verification methods have been restricted to less expressive temporal logics such as CTL [3, 17, 18, 38], CTL* [14], and linear-time logics such as LTL [16, 20, 31]. As already mentioned, the translation given in Section 3.1 can be considered a special case of the translation of Watanabe et al. [40] for higher-order programs. For imperative programs, however, our translation in Section 3.1 is more direct. Our translation for infinite-data programs may also be viewed as a generalization of Andersen's translation from modal μ-calculus model checking of finite-state systems to Boolean graphs [2].

The reduction from linear-time properties of first-order recursive programs to the validity checking problem in a first-order fixpoint logic is new, to our knowledge. Kobayashi et al. [26] proposed a translation from linear-time properties of higher-order programs to the validity checking in a *higher-order* fixpoint logic (called HFL), but their translation yields second-order fixpoint logic formulas for first-order recursive programs.[9] Combined with our Mu-Arithmetic prover, the translation yields a new automated method for proving linear-time properties of first-order recursive programs. Our translation may be considered a generalization of the technique for LTL model checking of recursive state machines [1], to deal with infinite-data programs.

Our approach of Mu2CHC described in Section 4 has been inspired by termination verification methods [21, 34] and generalizes the method of Fedyukovich et al. [21]. A related technique has been proposed by Biere et al. [4] for finite-state model checking. The Mu2CHC approach also much relies on the recent advance of CHC solving techniques [5, 13, 27, 37]. An alternative approach to approximate μ-formulas by ν-formulas would be to generalize the termination verification method based on transition invariants [35], as sketched in [40].

As discussed in Section 2.3, the validity checking problem for Mu-Arithmetic may be seen as a generalization of the satisfiability problem for CHC [8, 9]. A few extensions of CHC have been previously studied [3, 7]. To encode CTL verification problems, Beyene et al. [3] extended CHC with a special predicate *dwf* such that $dwf(r)$ if and only if r is disjunctively well-founded. This fragment is close to Mu-Arithmetic, in that, as an alternative to the method in Section 4, we can replace a μ-equation $P(\widetilde{x}) =_\mu \varphi$ with $P(\widetilde{x}) =_\nu \varphi'$, where φ' is the formula obtained from φ by replacing each subformula of the form $P(\widetilde{a})$ with $r(\widetilde{a}, \widetilde{x}) \wedge P(\widetilde{a})$ for a well-founded relation r. Allowing universal quantifiers in bodies of CHC [7] corresponds to allowing existential quantifiers in our HES (recall that we took

[9] They assume that source programs are in a CPS (continuation passing style) form, and then translate an order-n program to an order-n HFL formula. Since a first-order recursive program is converted to an order-2 CPS program, the order of the formula obtained by their translation is 2.

de Morgan dual in the conversions between a fragment of Mu-Arithmetic and CHC in Section 2.3). In contrast, there is no counterpart of the extension with existential quantifiers (in head positions of CHC) [3] in our fixpoint logic, which indicates that such an extension is unnecessary for the μ-calculus model checking of while programs.

7 Conclusion

We have proposed a method for proving validity of first-order fixpoint logic with integer arithmetic. Combined with the reduction in Section 3.1, the proposed methods yield an automated, unifying verification method for temporal properties of while-programs, supporting all the properties expressive in the modal μ-calculus. We have also presented a reduction from linear-time properties of first-order recursive programs to validity of fixpoint formulas, which also yields an automated method for temporal properties of first-order recursive programs, supporting all the properties expressive by Büchi automata. Future work includes further refinement of our verification method (e.g., on the point discussed in Remark 3), and an extension of our tool lo support data types other than integers. Extending our methods in Sections 3 and 4 to support algebraic data types is not difficult, but the CHC solving phase may become a bottleneck, as the current CHC solvers are not very good at dealing with algebraic data types.

We also plan to extend the methods to deal with *higher-order* fixpoint logic with integer arithmetic, so that temporal properties of higher-order functional programs can be automatically verified based on the work of Kobayashi et al. [26, 40].

Acknowledgments We would like to thank anonymous referees for useful comments. This work was supported by JSPS KAKENHI Grant Number JP15H05706 and JP16H05856.

References

1. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T.W., Yannakakis, M.: Analysis of recursive state machines. ACM Trans. Program. Lang. Syst. **27**(4), 786–818 (2005)
2. Andersen, H.R.: Model checking and boolean graphs. Theor. Comput. Sci. **126**(1), 3–30 (1994)
3. Beyene, T.A., Popeea, C., Rybalchenko, A.: Solving existentially quantified Horn clauses. In: CAV '13. LNCS, vol. 8044, pp. 869–882. Springer (2013)
4. Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. Electr. Notes Theor. Comput. Sci. **66**(2), 160–177 (2002)
5. Bjørner, N., Gurfinkel, A.: Property directed polyhedral abstraction. In: VMCAI '15. pp. 263–281. Springer (2015)
6. Bjørner, N., Gurfinkel, A., McMillan, K.L., Rybalchenko, A.: 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, vol. 9300, pp. 24–51. Springer (2015)

7. Bjørner, N., McMillan, K., Rybalchenko, A.: On solving universally quantified Horn clauses. In: SAS '13. LNCS, vol. 7935, pp. 105–125. Springer (2013)
8. Bjørner, N., McMillan, K.L., Rybalchenko, A.: Program verification as satisfiability modulo theories. In: 10th International Workshop on Satisfiability Modulo Theories, SMT 2012. pp. 3–11. EasyChair (2012)
9. Bjørner, N., McMillan, K.L., Rybalchenko, A.: Higher-order program verification as satisfiability modulo theories with algebraic data-types. CoRR **abs/1306.5264** (2013)
10. Blass, A., Gurevich, Y.: Existential fixed-point logic. In: Computation Theory and Logic, In Memory of Dieter Rödding. LNCS, vol. 270, pp. 20–36. Springer (1987)
11. Bradfield, J.C.: Fixpoint alternation and the game quantifier. In: CSL '99. LNCS, vol. 1683, pp. 350–361. Springer (1999)
12. Burn, T.C., Ong, C.L., Ramsay, S.J.: Higher-order constrained horn clauses for verification. PACMPL **2**(POPL), 11:1–11:28 (2018)
13. Champion, A., Chiba, T., Kobayashi, N., Sato, R.: ICE-based refinement type discovery for higher-order functional programs. In: TACAS '18. LNCS, vol. 10805, pp. 365–384. Springer (2018)
14. Cook, B., Khlaaf, H., Piterman, N.: Fairness for infinite-state systems. In: TACAS '15. pp. 384–398. Springer (2015)
15. Cook, B., Khlaaf, H., Piterman, N.: Verifying increasingly expressive temporal logics for infinite-state systems. J. ACM **64**(2), 15:1–15:39 (2017)
16. Cook, B., Koskinen, E.: Making prophecies with decision predicates. In: POPL '11. pp. 399–410. ACM (2011)
17. Cook, B., Koskinen, E.: Reasoning about nondeterminism in programs. In: PLDI '13. pp. 219–230. ACM (2013)
18. Cook, B., Koskinen, E., Vardi, M.: Temporal property verification as a program analysis task. In: CAV '11. pp. 333–348. Springer (2011)
19. Delzanno, G., Podelski, A.: Constraint-based deductive model checking. STTT **3**(3), 250–270 (2001)
20. Dietsch, D., Heizmann, M., Langenfeld, V., Podelski, A.: Fairness modulo theory: A new approach to LTL software model checking. In: Proceedings of CAV 2015. Lecture Notes in Computer Science, vol. 9206, pp. 49–66. Springer (2015)
21. Fedyukovich, G., Zhang, Y., Gupta, A.: Syntax-guided termination analysis. In: CAV '18. LNCS, vol. 10981, pp. 124–143. Springer (2018)
22. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: Proceedings of PLDI '12. pp. 405–416 (2012)
23. Hofmann, M., Chen, W.: Abstract interpretation from Büchi automata. In: CSL-LICS '14. pp. 51:1–51:10. ACM (2014)
24. Jaffar, J., Santosa, A.E., Voicu, R.: A CLP method for compositional and intermittent predicate abstraction. In: Proceedings of VMCAI 2006. Lecture Notes in Computer Science, vol. 3855, pp. 17–32. Springer (2006)
25. Kobayashi, N., Lozes, É., Bruse, F.: On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 246–259. ACM (2017)
26. Kobayashi, N., Tsukada, T., Watanabe, K.: Higher-order program verification via HFL model checking. In: ESOP '18. pp. 711–738. Springer (2018)
27. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. In: CAV '14. LNCS, vol. 8559, pp. 17–34. Springer (2014)
28. Koskinen, E., Terauchi, T.: Local temporal reasoning. In: CSL-LICS '14. pp. 59:1–59:10. ACM (2014)

29. Lozes, É.: A type-directed negation elimination. In: Matthes, R., Mio, M. (eds.) Proceedings Tenth International Workshop on Fixed Points in Computer Science, FICS 2015, Berlin, Germany, September 11-12, 2015. EPTCS, vol. 191, pp. 132–142 (2015)
30. Lubarsky, R.S.: μ-definable sets of integers. Journal of Symbolic Logic **58**(1), 291–313 (1993)
31. Murase, A., Terauchi, T., Kobayashi, N., Sato, R., Unno, H.: Temporal verification of higher-order functional programs. In: Proceedings of POPL 2016 (2016), to appear
32. Nanjo, Y., Unno, H., Koskinen, E., Terauchi, T.: A fixpoint logic and dependent effects for temporal property verification. In: LICS '18. pp. 759–768. ACM (Jul 2018)
33. Peralta, J.C., Gallagher, J.P., Saglam, H.: Analysis of imperative programs through analysis of constraint logic programs. In: Proceedings of SAS '98. Lecture Notes in Computer Science, vol. 1503, pp. 246–261. Springer (1998)
34. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS '04. pp. 32–41. IEEE (2004)
35. Podelski, A., Rybalchenko, A.: Transition invariants. In: 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings. pp. 32–41 (2004)
36. Seidl, H., Neumann, A.: On guarding nested fixpoints. In: Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1683, pp. 484–498. Springer (1999)
37. Unno, H., Torii, S., Sakamoto, H.: Automating induction for solving Horn clauses. In: CAV '17. pp. 571–591. Springer (2017)
38. Urban, C., Ueltschi, S., Müller, P.: Abstract interpretation of CTL properties. In: SAS '18. LNCS, vol. 11002, pp. 402–422. Springer (2018)
39. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: CONCUR. Lecture Notes in Computer Science, vol. 3170, pp. 512–528. Springer (2004)
40. Watanabe, K., Tsukada, T., Oshikawa, H., Kobayashi, N.: Reduction from branching-time property verification of higher-order programs to HFL validity checking. In: PEPM '19. ACM (2019)

Appendix

A Correctness of the Translation

This section proves the correctness of the translation in Section 3. Theorem 1 can be split into the following two theorems.

Theorem 2 (completeness). *Let D be a program and $\mathcal{A}$ be a Büchi automaton. If $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$, then $\Phi_D \models \mathbf{main}_{q_0,\mathtt{tt}}()$.*

Theorem 3 (soundness). *Let D be a program and $\mathcal{A}$ be a Büchi automaton. If $\Phi_D \models \mathbf{main}_{q_0,\mathtt{tt}}()$, then $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$.*

We later use the characterizations of least/greatest fixpoints by ordinals. Let $F \in D \to D$ be a monotonic function on complete lattice D. For each ordinal γ, we define F^γ by:

$$F^\gamma(x) = \begin{cases} x & \text{if } \gamma = 0 \\ F(F^{\gamma'}(x)) & \text{if } \gamma = \gamma' + 1 \\ \bigsqcup_{\gamma' < \gamma} F^{\gamma'}(x) & \text{if } \gamma \text{ is a limit ordinal.} \end{cases}$$

The following is a standard fact (one can choose as γ an ordinal greater than the cardinality of D).

Fact 4 *Let D be a complete lattice and $F \in D \to D$ a monotonic function. Then, there exists an ordinal γ such that $\mathbf{FP}_\mu(F) = F^\gamma(\bot_D)$ and $\mathbf{FP}_\nu(F) = F^\gamma(\top_D)$.*

We define the size of an expression e, written $\#(e)$, by

$$\#(v) = \#(*) = 1 \qquad \#(a_1 \,\mathbf{op}\, a_2) = \#(a_1) + \#(a_2)$$
$$\#(A; e) = 1 + \#(e) \qquad \#(f(v_1, \ldots, v_k)) = 2 + k$$
$$\#(\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2) = \#(\mathbf{if}\ v \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2) = \#(e_1) + \#(e_2) + 1$$

Below we first characterize the verification problem by product construction semantics in Section A.1. We then prove completeness and soundness in Sections A.2 and A.3 respectively.

A.1 Characterization of the Verification Problem by Product Construction Semantics

We introduce another operational semantics, which is convenient for characterizing the verification problem. Let D be a program and $\mathcal{A}$ be a Büchi automaton. We define the transition relation $(e, q) \xrightarrow{b}_D (e', q')$ by the following rules.

$$\frac{n \in Z}{(E[*], q) \xrightarrow{\mathtt{ff}}_D (E[n], q)}$$

$$\frac{a \notin Z \qquad \mathbf{val}(a) = n}{(E[a], q) \xrightarrow{\mathtt{ff}}_D (E[n], q)}$$

$$\frac{(f(x_1, \ldots, x_k) = e) \in D}{(E[f(n_1, \ldots, n_k)], q) \xrightarrow{\mathtt{ff}}_D (E[[n_1/x_1, \ldots, n_k/x_k]e], q)}$$

$$\frac{}{(E[\mathbf{let}\ x = n\ \mathbf{in}\ e], q) \xrightarrow{\mathtt{ff}}_D (E[[n/x]e], q)}$$

$$\frac{n \geq 0}{(E[\mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2], q) \xrightarrow{\mathtt{ff}}_D (E[e_1], q)}$$

$$\frac{n < 0}{(E[\mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2], q) \xrightarrow{\mathtt{ff}}_D (E[e_2], q)}$$

$$\frac{q' \in \Delta(q, A) \qquad b = (q' \in F)}{(E[A; e], q) \xrightarrow{b}_D (E[e], q')}$$

We often omit the subscript D. The relation $(e, q) \xrightarrow{b}_D (e', q')$ represents a transition of the product of the program D and the automaton $\mathcal{A}$, where the automaton may change its state from q to q' according to the event that may be generated by the transition of the program from e to e', and b represents whether the automaton has changed its state to a final state. We write $(e, q) \xRightarrow{b}_D (e', q')$ if $(e, q) \xrightarrow{b_1}_D \cdots \xrightarrow{b_n}_D (e', q')$ and $b = b_1 \vee \cdots \vee b_n$ for some $n \geq 0$ and $b_1, \ldots, b_n$.

The following lemma follows immediately from the definitions.

Lemma 1. *$\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if there exists an infinite transition sequence*

$$(\mathbf{main}(), q_0) \xrightarrow{b_1}_D (e_1, q_1') \xrightarrow{b_2}_D (e_2, q_2') \xrightarrow{b_3}_D \cdots$$

where $b_i = \mathtt{tt}$ for infinitely many i's.

A.2 Completeness

Suppose that $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$. By Lemma 1, there exists an infinite transition sequence

$$\pi = (\mathbf{main}(), q_0) \xrightarrow{b_1} (e_1, q_1) \xrightarrow{b_2} (e_2, q_2) \xrightarrow{b_3} \cdots$$

where $b_i = \mathtt{tt}$ for infinitely many i's.

We define ρ_π by:

$$\rho_\pi(f_{q,b,q'})(\widetilde{n},r) = \begin{cases} \mathtt{tt} & \text{if } (e_j,q_j) = (E[f(\widetilde{n})],q) \text{ and } (e_k,q_k) = (E[r],q') \text{ for some} \\ & j<k, \text{ with } b = b_{j+1} \vee \cdots \vee b_k \text{ and } e_i \text{ is of the form } E[e_i'] \\ & \text{for every } i \text{ such that } j \le i \le k, \text{ where } e_j' = f(\widetilde{n}) \xrightarrow{b_{j+1}}_D \\ & \cdots \xrightarrow{b_k}_D e_k' = r. \qquad \cdots (*1) \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

$$\rho_\pi(f_{q,b})(\widetilde{n}) = \begin{cases} \mathtt{tt} & \text{if } e_j = (E[f(\widetilde{n})]) \text{ for some } j, \text{ and the function call } f(\widetilde{n}) \\ & \text{never returns in } \pi. \qquad \cdots (*2) \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

Since $\rho_\pi(\mathbf{main}_{q_0,\mathtt{tt}})() = \mathtt{tt}$, Theorem 2 follows immediately from the following lemma.

Lemma 2. *$\rho_\pi \sqsubseteq [\![\Phi_D]\!]$.*

To prove Lemma 2, we prepare a few lemmas.

The following lemma (together with Lemma 8 proved later) ensures that the formula $[e]_{q,b,q',r}$ follows the intuition explained in Section 3.2.

Lemma 3. *If $(e,q) \stackrel{b}{\Longrightarrow}_D (r,q')$ and $\rho(f_{q_1,b',q_2})(\widetilde{n},r') = \mathtt{tt}$ for every subreduction[10] sequence of the form $(E[f(\widetilde{n})],q_1) \stackrel{b'}{\Longrightarrow}_D (E[r'],q_2)$, then $[\![[e]_{q,b,q',r}]\!]_\rho = \mathtt{tt}$.*

Proof. The proof proceeds by induction on the length of the reduction sequence $(e,q) \stackrel{b}{\Longrightarrow}_D (r,q')$, with case analysis on e.

- Case $e \equiv n \in Z$: In this case, $n = r$, $q = q'$ and $b = \mathtt{ff}$. Thus, $[e]_{q,b,q',r} = (n = r)$; hence the result follows immediately.
- Case $e \equiv a$ with $a \notin \mathbf{Z}$: In this case, $\mathbf{val}(a) = r$, $q = q'$ and $b = \mathtt{ff}$. Thus, $[\![[e]_{q,b,q',r}]\!]_\rho = [\![a = r]\!]_\rho = \mathtt{tt}$.
- Case $e \equiv *$: In this case, $q = q'$ and $b = \mathtt{ff}$. Thus, $[e]_{q,b,q',r} = \mathtt{tt}$.
- Case $e \equiv f(\widetilde{n})$: by the assumption, $\rho(f_{q,b,q'})(\widetilde{n},r) = \mathtt{tt}$.
- Case $e \equiv \mathbf{if}\ n \ge 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$: We discuss only the case where $n \ge 0$; the other case is similar. We have $(e,q) \xrightarrow{\mathtt{ff}}_D (e_1,q) \stackrel{b}{\Longrightarrow}_D (r,q')$. By the induction hypothesis, $[\![[e_1]_{q,b,q',r}]\!]_\rho = \mathtt{tt}$. Thus, $[\![[e]_{q,b,q',r}]\!]_\rho = [\![(n \ge 0) \wedge [e_1]_{q,b,q',r} \vee \cdots]\!]_\rho = \mathtt{tt}$.
- Case $e \equiv A; e'$: In this case, $(e,q) \xrightarrow{q'' \in F}_D (e',q'') \stackrel{b'}{\Longrightarrow}_D (r,q')$, with $b = (q'' \in F) \vee b'$ and $q'' \in \Delta(q,A)$. By the induction hypothesis, $[\![[e']_{q'',b',q',r}]\!]_\rho = \mathtt{tt}$. The result follows, since $[A;e']_{q,b',q',r} = [e']_{q'',b',q',r} \vee \cdots$.
- Case $e \equiv \mathbf{let}\ x = n\ \mathbf{in}\ e'$: In this case, $(e,q) \xrightarrow{\mathtt{ff}}_D ([n/x]e',q) \stackrel{b}{\Longrightarrow}_D (r,q')$, The result follows immediately from the induction hypothesis.
- Case $e \equiv \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$, with $e_1 \notin \mathbf{Z}$: We have:

$$(e,q) \xrightarrow{b_1}_D (\mathbf{let}\ x = r_1\ \mathbf{in}\ e_2, q'') \stackrel{b_2}{\Longrightarrow}_D (r,q'),$$

[10] Including $(e,q) \stackrel{b}{\Longrightarrow}_D (r,q')$ itself.

with $b = b_1 \vee b_2$ with $(e_1, q) \stackrel{b_1}{\Longrightarrow}_D (r_1, q'')$. By the induction hypothesis,

$$[\![[\mathbf{let}\ x = r_1\ \mathbf{in}\ e_2]_{q'',b_2,q',r}]\!]_\rho = [\![\exists x.(x = r_1 \wedge [e_2]_{q'',b_2,q',r})]\!]_\rho = \mathtt{tt},$$

and $[\![[e_1]_{q,b_1,q'',r_1}]\!]_\rho = \mathtt{tt}$. The result follows, since $[e]_{q,b,q',r} = \exists x.([e_1]_{q,b_1,q'',x} \wedge [e_2]_{q'',b_2,q',r})$, which is entailed by $\exists x.(x = r_1 \wedge [e_2]_{q'',b_2,q',r}) \wedge [e_1]_{q,b_1,q'',r_1}$. □

The next few lemmas state properties of the formula $[e]_{q,b}$.

Lemma 4. *If* $[\![[e]_{q,b}]\!]_\rho = \mathtt{tt}$*, then* $[\![[E[e]]_{q,b}]\!]_\rho = \mathtt{tt}$.

Proof. Straightforward induction on E. □

Lemma 5. *Let* $b \in \mathbf{B}$. *Suppose: (i)* $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$*, (ii)* $\rho(f_{q',b \vee b'})(\widetilde{n}) = \mathtt{tt}$*, and (iii) if* $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$ *contains a reduction of the form* $(E'[g(\widetilde{m})], q_1) \stackrel{\mathtt{ff}}{\longrightarrow}_D (E'[e'], q_1)$*, the reduction is extended to* $(E'[g(\widetilde{m})], q_1) \stackrel{\mathtt{ff}}{\Longrightarrow}_D (E'[r], q_1)$ *inside* $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$ *(i.e., every function call in* $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$ *returns inside it), and* $\rho(g_{q_1,b_1,q_2})(\widetilde{m}, r) = \mathtt{tt}$*. Then,* $[\![[e]_{q,b}]\!]_\rho = \mathtt{tt}$.

Proof. The proof proceeds by double induction on the length of the reduction sequence $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$ and the size of e. In the base case (where the length is 0), $e = E[f(\widetilde{n})]$. By the assumption (ii) and Lemma 4, we have $[\![[e]_{q,b}]\!]_\rho = \mathtt{tt}$ as required. For the induction step, we perform case analysis on e.

- Case $e \equiv a$ or $e \equiv *$: This contradicts the assumption $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$.
- Case $e \equiv g(\widetilde{m})$: This contradicts the assumption (iii) (that every function call returns).
- Case $e \equiv \mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$: We discuss only the case where $n \geq 0$; the other case is similar. We have $(e, q) \stackrel{\mathtt{ff}}{\longrightarrow}_D (e_1, q) \stackrel{b}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$. By the induction hypothesis, $[\![[e_1]_{q,b}]\!]_\rho = \mathtt{tt}$. Thus, $[\![[e]_{q,b}]\!]_\rho = [\![(n \geq 0) \wedge [e_1]_{q,b} \vee \cdots]\!]_\rho = \mathtt{tt}$.
- Case $e \equiv A; e'$: In this case, $(e, q) \stackrel{q'' \in F}{\longrightarrow}_D (e', q'') \stackrel{b''}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$, with $b' = (q'' \in F) \vee b''$ and $q'' \in \Delta(q, A)$. By the induction hypothesis, $[\![[e']_{q'',b \vee (q'' \in F)'}]\!]_\rho = \mathtt{tt}$. The result follows, since $[A; e']_{q,b} = [e']_{q'',b \vee (q'' \in F)} \vee \cdots$.
- Case $e \equiv \mathbf{let}\ x = n\ \mathbf{in}\ e'$: In this case, $(e, q) \stackrel{\mathtt{ff}}{\longrightarrow}_D ([n/x]e', q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$. The result follows immediately from the induction hypothesis.
- Case $e \equiv \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$, with $e_1 \notin \mathbf{Z}$: If e_1 evaluates to an integer in the reduction sequence $(e, q) \stackrel{b'}{\Longrightarrow}_D (E[f(\widetilde{n})], q')$, i.e., if

$$(e, q) \stackrel{b_1}{\longrightarrow}_D (\mathbf{let}\ x = r_1\ \mathbf{in}\ e_2, q'') \stackrel{b_2}{\Longrightarrow}_D (E[f(\widetilde{n})], q'),$$

then, by the induction hypothesis,

$$[\![[\mathbf{let}\ x = r_1\ \mathbf{in}\ e_2]_{q'',b\vee b_1}]\!]_\rho = [\![\exists x.(x = r_1 \wedge [e_2]_{q'',b\vee b_1})]\!]_\rho = \mathtt{tt}.$$

By Lemma 3, we also have $[\![[e_1]_{q,b_1,q'',r_1}]\!]_\rho = \mathtt{tt}$. The result follows, since $[e]_{q,b} = \cdots \vee \exists x.([e_1]_{q,b_1,q'',x} \wedge [e_2]_{q'',b\vee b_1})$, which is entailed by $(x = r_1 \wedge [e_1]_{q,b_1,q'',x} \wedge [e_2]_{q'',b\vee b_1})$.

If e_1 does not evaluate to an integer in the reduction sequence, $E = \mathbf{let}\ x = E_1\ \mathbf{in}\ e_2$. By the induction hypothesis, $[\![[e_1]_{q,b}]\!]_\rho = \mathtt{tt}$. By Lemma 4, we have $[\![[e]_{q,b}]\!]_\rho = \mathtt{tt}$ as required. □

The following is an (co)induction principle about nexted fixpoints, which is used for proving Lemma 2.

Lemma 6 (mixed induction). *Let D_1, D_2 be complete lattices. Assume that $F_2 : D_2 \times D_1 \to D_2, F_1 : D_2 \times D_1 \to D_1$ are monotonic functions, and that*

$$x_2 = \mathbf{FP}_\nu(\lambda z_2.F_2(\mathbf{FP}_\mu(z_2, \lambda z_1.F_1(z_2, z_1)))) \qquad x_1 = \mathbf{FP}_\mu(\lambda z_1.F_1(x_2, z_1)).$$

If $y_1 \in D_1$ and $y_2 \in D_2$ satisfy:

$$y_2 \sqsubseteq F_2(y_2, y_1) \qquad y_1 \sqsubseteq \mathbf{FP}_\mu(\lambda z_1.F_1(y_2, z_1)),$$

then $y_1 \sqsubseteq x_1$ and $y_2 \sqsubseteq x_2$.

Proof. By the assumption, we have:

$$y_2 \sqsubseteq F_2(y_2, y_1) \sqsubseteq F_2(y_2, \mathbf{FP}_\mu(\lambda z_1.F_1(y_2, z_1))),$$

i.e., y_2 is a fixpoint of $\lambda z_2.F_2(\mathbf{FP}_\mu(z_2, \lambda z_1.F_1(z_2, z_1)))$. We have thus $y_2 \sqsubseteq x_2$. Using the assumption and $y_2 \sqsubseteq x_2$, we have

$$y_1 \sqsubseteq \mathbf{FP}_\mu(\lambda z_1.F_1(y_2, z_1)) \sqsubseteq \mathbf{FP}_\mu(\lambda z_1.F_1(x_2, z_1)) = x_1,$$

as required. □

We are now ready to prove Lemma 2.

Proof (Lemma 2). Let ρ_1 and ρ_2 be the restrictions of ρ_π to $\mathcal{P}_{D,\mathtt{fin}} \cup \mathcal{P}_{D,\mathtt{ff}}$ and $\mathcal{P}_{D,\mathtt{tt}}$ respectively. By Lemma 6 and the definition of $[\![\Phi_D]\!]$, it suffices to show:

$$\rho_2 \sqsubseteq [\![\mathcal{E}_{D,\mathtt{tt}}]\!]_{\rho_\pi} \tag{1}$$

$$\rho_1 \sqsubseteq [\![\Phi_D]\!]_1(\rho_2) (= \mathbf{FP}_\mu(\lambda\rho \in \mathit{\Gamma}_{\mathcal{P}_{D,\mathtt{fin}}\cup\mathcal{P}_{D,\mathtt{ff}}}.[\![\mathcal{E}_{D,\mathtt{tt}}]\!]_{\rho_2\cup\rho})) \tag{2}$$

To show (2), we define the ranks of $(f_{q,b,q'}, \widetilde{n}, r)$ and $(f_{q,b}, \widetilde{n})$ by:

- $\mathtt{rank}_\pi(f_{q,b,q'}, \widetilde{n}, r)$ is the least value of $k - j$ for j and k that satisfy the condition (*1) (in the definition of ρ_π).
- $\mathtt{rank}_\pi(f_{q,b}, \widetilde{n})$ is the least value of $k - j$, where j satisfies the condition (*2) (in the definition of ρ_π) and k is the least value greater than j such that $b_k = \mathtt{tt}$.

We show

(i) $\rho_\pi(f_{q,b,q'})(\widetilde{n}, r) = \mathtt{tt}$ implies $(\llbracket \Phi_D \rrbracket_1(\rho_2))(f_{q,b,q'})(\widetilde{n}, r) = \mathtt{tt}$.
(ii) $\rho_\pi(f_{q,\mathtt{ff}})(\widetilde{n}) = \mathtt{tt}$ implies $(\llbracket \Phi_D \rrbracket_1(\rho_2))(f_{q,b})(\widetilde{n}) = \mathtt{tt}$.

by induction on $\mathtt{rank}_\pi(f_{q,b,q'}, \widetilde{n}, r)$ and $\mathtt{rank}_\pi(f_{q,b}, \widetilde{n})$ respectively; that will complete the proof of (2). The former is also used for proving the latter.

We first show (i) by induction on $\mathtt{rank}_\pi(f_{q,b,q'}, \widetilde{n}, r)$. By the definition of ρ_π and $\mathtt{rank}_\pi$, there exist j and k such that $k - j = \mathtt{rank}_\pi(f_{q,b,q'}, \widetilde{n}, r)$, and

$$(e_j, q_j) = (E[f(\widetilde{n})], q) \xrightarrow{\mathtt{ff}}_D (E[[\widetilde{n}/\widetilde{x}]e_f], q) \xRightarrow{b}_D (E[r], q') = (e_k, q_k).$$

By the induction hypothesis, $(\llbracket \Phi_D \rrbracket_1(\rho_2)))(g_{q_1,b',q_2})(\widetilde{m}, r') = \mathtt{tt}$ for every subreduction sequence $(E_1[g(\widetilde{m})], q_1) \xRightarrow{b'}_D (E_1[r'], q_2)$ of $(E[[\widetilde{n}/\widetilde{x}]e_f], q) \xRightarrow{b}_D (E[r], q')$. Thus, by Lemma 3,

$$(\llbracket \Phi_D \rrbracket_1(\rho_2))(f_{q,b,q'})(\widetilde{n}, r) = \llbracket [\widetilde{n}/\widetilde{x}]e_f \rrbracket_{q,b,q',r\,\rho_2 \cup (\llbracket \Phi_D \rrbracket_1(\rho_2))} = \mathtt{tt},$$

where $f(\widetilde{x}) = e_f \in D$, as required.

Next, we show (ii) by induction on $\mathtt{rank}_\pi(f_{q,b}, \widetilde{n})$. By the assumption $\rho_\pi(f_{q,b})(\widetilde{n}) = \mathtt{tt}$, π contains an infinite sequence $(E[f(\widetilde{n})], q) \xrightarrow{b'_1}_D (E[[\widetilde{n}/\widetilde{x}]e], q) \xrightarrow{b'_2}_D \cdots$, where $f(\widetilde{n})$ never returns, and $b'_{\mathtt{rank}_\pi(f_{q,b},\widetilde{n})} = \mathtt{tt}$ and $b'_i = \mathtt{ff}$ for every $i < \mathtt{rank}_\pi(f_{q,b}, \widetilde{n})$. Since $f(\widetilde{n})$ does not return, $(E[[\widetilde{n}/\widetilde{x}]e], q) \xrightarrow{b'_2}_D \cdots$ must be of the form: $(E[[\widetilde{n}/\widetilde{x}]e], q) \xRightarrow{b''}_D (E[E'[g(\widetilde{m})]], q')$ where g directly comes from e (in other words, $g(\widetilde{m})$ is a direct call from $f(\widetilde{n})$) and $g(\widetilde{m})$ does not return either. By the construction of ρ_π, $\rho_2(g_{q',\mathtt{tt}})(\widetilde{m}) = \mathtt{tt}$. Also, if $b'' = \mathtt{ff}$, then by the induction hypothesis, $(\llbracket \Phi_D \rrbracket_1(\rho_2))(g_{q',\mathtt{ff}})(\widetilde{m}) = \mathtt{tt}$. Thus, $([\widetilde{n}/\widetilde{x}]e, q) \xRightarrow{b''}_D (E'[g(\widetilde{m})], q')$ with $\rho = \rho_2 \cup (\llbracket \Phi_D \rrbracket_1(\rho_2))$ satisfies the assumptions of Lemma 5. Therefore, we have

$$(\llbracket \Phi_D \rrbracket_1(\rho_2))(f_{q,b})(\widetilde{n}) = \llbracket [\widetilde{n}/\widetilde{x}]e \rrbracket \rho_2 \cup (\llbracket \Phi_D \rrbracket_1(\rho_2)) = \mathtt{tt},$$

as required.

It remains to show (1). What we need to show is:

(iii) $\rho_\pi(f_{q,\mathtt{tt}})(\widetilde{n}) = \mathtt{tt}$ implies $\llbracket [\widetilde{n}/\widetilde{x}]e \rrbracket_{\rho_2 \cup (\llbracket \Phi_D \rrbracket_1(\rho_2))} = \mathtt{tt}$,

for every $(f(\widetilde{x}) = e) \in D$. By the assumption $\rho_\pi(f_{q,\mathtt{tt}})(\widetilde{n}) = \mathtt{tt}$, π contains an infinite reduction sequence

$$(E[f(\widetilde{n})], q) \xrightarrow{\mathtt{ff}}_D (E[[\widetilde{n}/\widetilde{x}]e], q) \xRightarrow{b_1}_D (E[E'[g(\widetilde{n})]], q') \xRightarrow{b_2}_D \cdots,$$

where $g(\widetilde{n})$ is a direct call from $f(\widetilde{n})$, and $g(\widetilde{n})$ never returns. The reduction sequence $([\widetilde{n}/\widetilde{x}]e, q) \xRightarrow{b_1}_D (E'[g(\widetilde{n})], q')$ with $\rho = \rho_2 \cup (\llbracket \Phi_D \rrbracket_1(\rho_2))$ satisfies the assumptions of Lemma 5. Thus, we have $\llbracket [\widetilde{n}/\widetilde{x}]e \rrbracket_{\rho_2 \cup (\llbracket \Phi_D \rrbracket_1(\rho_2))} = \mathtt{tt}$, as required. □

Theorem 2 follows immediately from Lemma 2.

Proof (Proof of Theorem 2). Suppose $\mathcal{L}(D) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$. By Lemma 1, there exists an infinite transition sequence

$$\pi = (\mathbf{main}(), q_0) \xrightarrow{b_1} (e_1, q_1) \xrightarrow{b_2} (e_2, q_2) \xrightarrow{b_3} \cdots$$

where $b_i = \mathtt{tt}$ for infinitely many i's. By the definition of ρ_π, $\rho_\pi(\mathbf{main}_{q_0,\mathtt{tt}})() = \mathtt{tt}$. By Lemma 2, we have $[\![\Phi_D]\!](\mathbf{main}_{q_0,\mathtt{tt}})() = \mathtt{tt}$, i.e., $\Phi_D \models \mathbf{main}_{q_0,\mathtt{tt}}()$. □

A.3 Soundness

We prove soundness (Theorem 3) in this section. Below, we write Θ for $\lambda\sigma_2 \in \Gamma_{\mathcal{P}_{D,\mathtt{tt}}}.\lambda\sigma_1 \in \Gamma_{\mathcal{P}_{D,\mathtt{fin}} \cup \mathcal{P}_{D,\mathtt{ff}}}.[\![\mathcal{E}_{\mathtt{fin}} \cup \mathcal{E}_{D,\mathtt{ff}}]\!]_{\sigma_1 \cup \sigma_2}$. We first prepare a few lemmas.

Lemma 7. *If $(\Theta(\sigma_2))^\gamma(\rho)(f)(\widetilde{n})) = \mathtt{tt}$ and $\rho(f)(\widetilde{n}) = \mathtt{ff}$, then there exists a successor ordinal γ' such that $(\Theta(\sigma_2))^{\gamma'}(\rho)(f)(\widetilde{n})) = \mathtt{tt}$ and $\gamma' \leq \gamma$.*

Proof. Since $\rho(f)(\widetilde{n}) = \mathtt{ff}$, γ must be a successor ordinal or a limit ordinal. If γ is a limit ordinal,

$$(\Theta(\sigma_2))^\gamma(\rho)(f)(\widetilde{n})) = \bigsqcup_{\gamma' < \gamma} (\Theta(\sigma_2))^{\gamma'}(\rho)(f)(\widetilde{n})).$$

Therefore, there exists γ' such that $(\Theta(\sigma_2))^{\gamma'}(\rho)(f)(\widetilde{n})) = \mathtt{tt}$. □

Let σ_0, σ_1, and σ_2 be the restrictions of $[\![\Phi_D]\!]$ to $\mathcal{P}_{D,\mathtt{fin}}$, $\mathcal{P}_{D,\mathtt{ff}}$ and $\mathcal{P}_{D,\mathtt{tt}}$ respectively. We write ρ_γ for $\sigma_2 \cup (\Theta(\sigma_2))^\gamma(\sigma_0)$.

Lemma 8. *If $[\![[e]_{q,b,q',r}]\!]_{\rho_\gamma} = \mathtt{tt}$, then $(e, q) \xrightarrow{b}_D (r, q')$.*

Proof. For an ordinal ξ, let ρ'_ξ be $(\Theta(\sigma_2))^\xi(\emptyset)$. Since $[e]_{q,b,q',r}$ contains only predicates of the form $f_{q,b,q'}$, it suffices to sow:

If $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$, then $(e, q) \xrightarrow{b}_D (r, q')$.

for any ξ. We prove it by double induction on ξ and the size of e.

- Case $e = a$: By the assumption $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$, we have $b = \mathtt{ff}$, $q = q'$, and $[\![a = r]\!]_{\rho'_\xi} = \mathtt{tt}$. Thus, the result follows immediately.
- Case $e = *$: By the assumption $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$, we have $b = \mathtt{ff}$ and $q = q'$. The result follows immediately, as $(*, q) \xrightarrow{\mathtt{ff}}_D (r, q)$.
- Case $e = f(\widetilde{n})$: By Lemma 7, we may assume that ξ is a successor ordinal. By $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$, we have $[\![[[\widetilde{n}/\widetilde{x}]/e_f]_{q,b,q',r}]\!]_{\rho'_{\xi-1}} = \mathtt{tt}$. Thus, the result follows from the induction hypothesis.
- Case $e = \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$: By the assumption $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$, there exists r_1, b_1, b_2, q'' such that $[\![[e_1]_{q,b_1,q'',r_1}]\!]_{\rho'_\xi} = \mathtt{tt}$ and $[\![[[r_1/x]e_2]_{q'',b_2,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$ with $b = b_1 \vee b_2$. By the induction hypothesis, we have $(e_1, q) \xRightarrow{b_1}_D (r', q'')$ and $([r/x]e_2, q'') \xRightarrow{b_2}_D (r, q')$. Thus, we have $(e, q) \xRightarrow{b}_D (r, q')$ as required.

- Case $e = \mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$: We discuss only the case $n \geq 0$, since the other case is similar. Since $[e]_{q,b,q',r} = ((n \geq 0) \wedge [e_1]_{q,b,q',r}) \vee ((n < 0) \wedge [e_1]_{q,b,q',r})$, by the assumptions $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$ and $n \geq 0$, we have $[\![[e_1]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$. By the induction hypothesis, we have $(e_1, q) \overset{b}{\Longrightarrow}_D (r, q')$, from which the result follows immediately.
- Case $e = A; e'$: By the assumption $[\![[e]_{q,b,q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$, there exists b' and $q'' \in \Delta(q, A)$ such that $[\![[e']_{q'',b',q',r}]\!]_{\rho'_\xi} = \mathtt{tt}$ and $b = (q'' \in F) \vee b'$. By the induction hypothesis, we have $(e', q'') \overset{b'}{\Longrightarrow}_D (r, q')$, from which the result follows immediately. □

Lemma 9. *If* $[\![[e]_{q,b}]\!]_{\rho_\gamma} = \mathtt{tt}$*, then either (i)* $e = E[f(\widetilde{n})]$ *and* $\rho_\gamma(f_{q,b})(\widetilde{n}) = \mathtt{tt}$*, or (ii) there exists* e' *such that* $(e, q) \overset{b'}{\Longrightarrow}_D (e', q')$ *and* $\#(e') < \#(e)$ *and* $[\![[e']_{q',b\vee b'}]\!]_{\rho_\gamma} = \mathtt{tt}$.

Proof. Since $[\![[e]_{q,b}]\!]_{\rho_\gamma} = \mathtt{tt}$, e is not a value. So, e must be of the form $e = E[I]$. We show the required property by induction on E. For the base case (where $E = [\,]$), we perform case analysis on I:

- Case I is a or $*$: This contradicts the assumption $[\![[e]_{q,b}]\!]_{\rho_\gamma} = \mathtt{tt}$.
- Case $I = f(\widetilde{n})$: (i) holds.
- Case $I = \mathbf{let}\ x = n\ \mathbf{in}\ e_1$: In this case, we have: $[e]_{q,b} = \exists x.x = n \wedge [e_1]_{q,b}$, which is logically equivalent to $[n/x][e_1]_{q,b} = [[n/x]e_1]_{q,b}$. Thus, the required result holds for $e' = [n/x]e_1$ and $q' = q$.
- Case $I = \mathbf{if}\ n \geq 0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$: We discuss only the case $n \geq 0$; the other case $n < 0$ is similar. If $n \geq 0$, we have

$$[\![[e]_{q,b}]\!]_{\rho_\gamma} = [\![((n \geq 0) \wedge [e_1]_{q,b}) \vee ((n < 0) \wedge [e_2]_{q,b})]\!]_{\rho_\gamma} = [\![[e_1]_{q,b}]\!]_{\rho_\gamma}.$$

  Thus, the required result holds for $e' = e_1$ and $q' = q$.
- Case $I = A; e_1$: In this case, we have: $[e]_{q,b} = \exists q' \in \Delta(q, A).[e_1]_{q',b\vee b'}$. By the assumption $[\![[e]_{q,b}]\!]_{\rho_\gamma} = \mathtt{tt}$, there exists $q' \in \Delta(q, A)$ such that $[\![[e_1]_{q',b\vee b'}]\!]_{\rho_\gamma} = \mathtt{tt}$. Thus, the required result holds for $e' = e_1$.

For the induction step, suppose $E \equiv \mathbf{let}\ x = E_1\ \mathbf{in}\ e_2$. Since we have

$$[e]_{q,b} = [E_1[I]]_{q,b} \vee \exists x, b', q''.([E_1[I]]_{q,b',q'',x} \wedge [e_2]_{q'',b\vee b'}),$$

we have either $[\![[E_1[I]]_{q,b}]\!]_{\rho_\gamma} = \mathtt{tt}$, or there exists r, b', q'' such that $[\![[E_1[I]]_{q,b',q'',r}]\!]_{\rho_\gamma} = \mathtt{tt}$ and $[\![[[r/x]e_2]_{q'',b\vee b'}]\!]_{\rho_\gamma}$. In the former case, the result holds for $e' = E_1[I]$. In the latter case, by Lemma 8 and $[\![[E_1[I]]_{q,b',q'',r}]\!]_{\rho_\gamma} = \mathtt{tt}$, we have $(E_1[I], q) \overset{b'}{\Longrightarrow}_D (r, q'')$. Thus, the required result holds for $e' = [r/x]e_2$. □

Lemma 10. *If* $[\![[e]_{q,b}]\!]_{\rho_\gamma} = \mathtt{tt}$*, then* $(e, q) \overset{b'}{\Longrightarrow}_D (E[g(\widetilde{m})], q')$ *and* $[\![[g(\widetilde{m})]_{q,b\vee b'}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$ *for some* E*,* g*,* q*, and* $\widetilde{m}$.

Proof. This follows by straightforward induction on $\#(e)$, using Lemma 9.

Lemma 11. *If* $[\![[f(\widetilde{n})]_{q,\mathtt{tt}}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$*, then* $(f(\widetilde{n}), q) \stackrel{b}{\Longrightarrow}_D (E[g(\widetilde{m})], q')$ *and* $[\![[g(\widetilde{m})]_{q,b}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$ *for some* E*,* g*,* q*, and* $\widetilde{m}$*.*

Proof. By the assumption $[\![[f(\widetilde{n})]_{q,\mathtt{tt}}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$, we have $(f(\widetilde{n}), q) \xrightarrow{\mathtt{ff}}_D ([\widetilde{n}/\widetilde{x}]e, q)$ with $[\![[[\widetilde{n}/\widetilde{x}]e]_{q,\mathtt{ff}}]\!]_{[\![\Phi]\!]}$. By Lemma 10 (note that $\rho_\gamma = [\![\Phi]\!]$ for some ordinal γ), we have the required result. □

Lemma 12. *If* $[\![[f(\widetilde{n})]_{q,\mathtt{ff}}]\!]_{\rho_\gamma} = \mathtt{tt}$*, then* $(f(\widetilde{n}), q) \stackrel{b}{\Longrightarrow}_D (E[g(\widetilde{m})], q')$ *and* $[\![[g(\widetilde{m})]_{q',b}]\!]_{\rho_{\gamma'}} = \mathtt{tt}$*, for some* $\gamma' < \gamma$*.*

Proof. By the assumption $[\![[f(\widetilde{n})]_{q,\mathtt{ff}}]\!]_{\rho_\gamma} = \mathtt{tt}$, $(\Theta(\sigma_2))^\gamma(\sigma_0)(f_{q,\mathtt{ff}})(\widetilde{n}) = \mathtt{tt}$. By Lemma 7, we have a successor ordinal $\gamma'' \leq \gamma$ such that $(\Theta(\sigma_2))^{\gamma''}(\sigma_0)(f_{q,\mathtt{ff}})(\widetilde{n}) = \mathtt{tt}$. Thus, we have $(f(\widetilde{n}), q) \xrightarrow{\mathtt{ff}}_D ([\widetilde{n}/\widetilde{x}]e, q)$ with $[\![[[\widetilde{n}/\widetilde{x}]e]_{q,\mathtt{ff}}]\!]_{\rho_{\gamma''-1}}$. By Lemma 10, we have the required result for $\gamma' = \gamma'' - 1$. □

As a corollary, we have:

Lemma 13. *If* $[\![[f(\widetilde{n})]_{q,b}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$*, then* $(f(\widetilde{n}), q) \stackrel{\mathtt{tt}}{\Longrightarrow}_D (E[g(\widetilde{m})], q')$ *and* $[\![[g(\widetilde{m})]_{q,b'}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$ *for some* E*,* g*,* q*,* b'*, and* $\widetilde{m}$*.*

Proof. By Lemmas 11 and 12, we have either (i) $(f(\widetilde{n}), q) \stackrel{\mathtt{tt}}{\Longrightarrow}_D (E[g(\widetilde{m})], q')$ and $[\![[g(\widetilde{m})]_{q,\mathtt{tt}}]\!]_{[\![\Phi]\!]} = \mathtt{tt}$, or (ii) $(f(\widetilde{n}), q) \stackrel{\mathtt{ff}}{\Longrightarrow}_D (E[g(\widetilde{m})], q')$ and $[\![[g(\widetilde{m})]_{q,\mathtt{ff}}]\!]_{\rho_\gamma} = \mathtt{tt}$ for some γ. In the latter case, by induction on γ and Lemma 12, we have the required result. □

Proof (Theorem 3). An immediate corollary of Lemma 13. □

B Additional Information about Benchmarks and Experiments

We provide additional information on our own benchmark set 1.

- The problems 1–6 are validity checking problems for the following fixpoint logic formulas, which do not necessarily come from program verification problems.

  $1 : \forall n.p_1(n) \Rightarrow n \geq 0$
  where $\{p_1(x) =_\nu p_2(x) \wedge p_1(x+1)\}; \{p_2(y) =_\mu y = 0 \vee p_2(y-1)\}$.
  $2 : \forall n.n \geq 0 \Rightarrow p_1(n)$ for p_1 above
  $3 : \forall a, b.\exists x \geq 0.\forall x'.x > x' \Rightarrow 2x' + a > x' + b$,
  where $\exists x$ and $\forall x'$ are encoded based on Remark 1.
  $4 - 6$: variations of 3

- Problem 7 is $\Phi_{D_0,\mathcal{A}_0}$ in Section 3.2, and 8 is its variation, obtained by replacing $\exists x.\mathtt{f}_{q_A,\mathtt{ff}}(x)$ with $\forall x.\mathtt{f}_{q_A,\mathtt{ff}}(x)$ in the body of $\mathbf{main}_{q_A,\mathtt{tt}}$.
- Problems 9 and 10 are from [23].

- Problems 11-13 are are from [28]; they correspond to the first three examples in Figure 10 of [28], except that the original version of koskinen1-fo is a higher-order program, and we have applied inlining to obtain an equivalent first-order program.
- Problem 14 is from [31].
- Problem 15 encodes the property that, for all m, n, event A occurs infinitely often in the following program:

```
let rec f n m =
  if n < m then
    A; f (n + 1) m
  else
    f n (m + 1)
in f n m
```

- Problems 16–19 are variations of Problem 15.
- Problem 20 is obtained from the property that, for every n, event A occurs infinitely often in the following program:

```
let rec f x = if x = 0 then 3 else f (x - 3) in
let rec g x =
  if x >= 0 then
    A; let a = f x in g (x + a)
  else loop()
in g(n)
```

  Problems 21 and 22 are variations of 20.
- Problems 23–25 are from Examples 4–6 in Appendix 3.1.
- Problems 26–28 encode CTL properties of labeled transition systems. The transition systems for 27 and 28 are quite tricky. The formula checked in 27 is $\forall n.E(n)$ where

$$\begin{array}{l}\{E(n) =_\mu V_2(n)\};\\ \{V_2(n) =_\nu V_3(n+1), V_3(n) =_\nu V_1(n+1),\\ \ V_1(n) =_\nu (n \leq 0 \wedge V_4(n+1)) \vee E(n-3),\\ \ V_4(n) =_\nu (n = 0 \wedge V_3(n-6)) \vee V_2(n+1)\},\end{array}$$

  and the formula for 28 is a variation of it, obtained by replacing the constant 6 with 5.

The results for the "small" benchmark set from [17] is shown in Table 2.

Table 2. Benchmark on all the small problems from [17].

Property	$\models \varphi$					$\models \neg\varphi$				
	Exp.	PLDI'13		Mu2CHC		Exp.	PLDI'13		Mu2CHC	
		Act.	Time[s]	Act.	Time[s]		Act.	Time[s]	Act.	Time[s]
1. AFp	✓	✓	1.2	✓	0.14	✗	✗	0.8	✗	0.16
2. AFp	✗	✗	0.8	✗	0.08	✓	✓	0.9	✓	0.09
3. AGp	✓	✓	0.3	✓	0.08	✗	✗	0.5	✗	0.08
4. AGp	✗	✗	0.5	✗	0.08	✓	✓	0.6	✓	0.08
5. EFp	✓	✓	-	✓	0.09	✗	✗	0.8	✗	0.09
6. EFp	✗	✗	1.7	✗	0.08	✓	✓	2.0	✓	0.08
7. EGp	✓	✓	0.9	✓	0.09	✗	✗	0.5	✗	0.09
8. EGp	✗	✗	0.9	✗	0.09	✓	✓	0.4	✓	0.06
9. AGAFp	✓	✓	10.8	✓	0.16	✗	✗	1.9	✗	0.18
10. AGAFp	✗	✗	1.9	✗	0.09	✓	✓	3.6	✓	0.10
11. AGEFp	✓	✓	29.0	✓	2.32	✗	✗	3.9	✗	3.44
12. AGEGp	✓	✓	1.2	✓	0.08	✗	✗	6.3	✗	0.09
13. AFEGp	✓	✓	55.8	✓	0.76	✗	✗	10.9	✗	1.03
14. AFEFp	✓	✓	3.7	✓	0.68	✗	✗	33.7	✗	0.18
15. AFAGp	✓	✓	1.3	✓	0.60	✗	✗	2.7	✗	0.82
16. AFAGp	✗	✗	11.0	✗	0.71	✓	✓	5.8	✓	0.18
17. EFEGp	✓	✓	44.3	✓	0.64	✗	✗	3.6	✗	0.96
18. EFEGp	✗	✗	54.7	✗	0.30	✓	✓	10.2	✓	1.53
19. EFAGp	✓	✓	0.6	✓	1.47	✗	✗	23.8	✗	0.13
20. EFAFp	✓	?	-	✓	0.33	✗	✗	7.5	✗	0.18
21. EGEFp	✗	✗	10.4	✗	0.29	✓	✓	40.3	✓	0.29
22. EGAGp	✓	✓	0.8	✓	0.06	✗	✗	0.9	✗	0.06
23. EGAFp	✓	✓	12.5	✓	0.39	✗	✗	13.8	✗	0.12
24. EG($q \Rightarrow$ EFp)	✓	✗	33.9	✓	0.57	✗	✗	2.0	✗	1.04
25. EG($q \Rightarrow$ AFp)	✗	✗	150.2	✗	0.08	✓	✗	13.8	✓	0.15
26. AG($q \Rightarrow$ EGp)	✓	✓	2.2	✓	0.09	✗	✗	6.3	✗	0.09
27. AG($q \Rightarrow$ EFp)	✓	✓	29.5	✓	1.46	✗	✗	3.9	✗	4.44