Higher-Order Program Verification via HFL Model Checking

Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe

The University of Tokyo

Abstract. There are two kinds of higher-order extensions of model checking: HORS model checking and HFL model checking. Whilst the former has been applied to automated verification of higher-order functional programs, applications of the latter have not been well studied. In the present paper, we show that various verification problems for functional programs, including may/must-reachability, trace properties, and linear-time temporal properties (and their negations), can be naturally reduced to (extended) HFL model checking. The reductions yield a sound and complete logical characterization of those program properties. Compared with the previous approaches based on HORS model checking, our approach provides a more uniform, streamlined method for higher-order program verification.

1 Introduction

There are two kinds of higher-order extensions of model checking in the literature: HORS model checking [17, 32] and HFL model checking [43]. The former is concerned about whether the tree generated by a given higher-order tree grammar called a higher-order recursion scheme (HORS) satisfies the property expressed by a given modal μ-calculus formula (or a tree automaton), and the latter is concerned about whether a given finite state system satisfies the property expressed by a given formula of higher-order modal fixpoint logic (HFL), a higher-order extension of the modal μ-calculus. Whilst HORS model checking has been applied to automated verification of higher-order functional programs [18, 19, 23, 26, 33, 42, 44], there have been few studies on applications of HFL model checking to program/system verification. Despite that HFL has been introduced more than 10 years ago, we are only aware of applications to assume-guarantee reasoning [43] and process equivalence checking [28].

In the present paper, we show that various verification problems for higher-order functional programs can actually be reduced to (extended) HFL model checking in a rather natural manner. We briefly explain the idea of our reduction below.[1] We translate a program to an HFL formula that says "the program has a valid behavior" (where the *validity* of a behavior depends on each verification problem). Thus, a program is actually mapped to a *property*, and a program

[1] In this section, we use only a fragment of HFL that can be expressed in the modal μ-calculus. Some familiarity with the modal μ-calculus [25] would help.

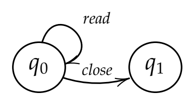


Fig. 1. File access protocol

property is mapped to a system to be verified; this has been partially inspired by the recent work of Kobayashi et al. [20], where HORS model checking problems have been translated to HFL model checking problems by switching the roles of models and properties.

For example, consider a simple program fragment $\mathtt{read}(x); \mathtt{close}(x)$ that reads and then closes a file (pointer) x. The transition system in Figure 1 shows a valid access protocol to read-only files. Then, the property that a read operation is allowed in the current state can be expressed by a formula of the form $\langle \mathtt{read} \rangle \varphi$, which says that the current state has a `read`-transition, after which φ is satisfied. Thus, the program $\mathtt{read}(x); \mathtt{close}(x)$ being valid is expressed as $\langle \mathtt{read} \rangle \langle \mathtt{close} \rangle \mathbf{true}$,[2] which is indeed satisfied by the initial state q_0 of the transition system in Figure 1. Here, we have just replaced the operations `read` and `close` of the program with the corresponding modal operators $\langle \mathtt{read} \rangle$ and $\langle \mathtt{close} \rangle$. We can also naturally deal with branches and recursions. For example, consider the program $\mathtt{close}(x) \Box (\mathtt{read}(x); \mathtt{close}(x))$, where $e_1 \Box e_2$ represents a non-deterministic choice between e_1 and e_2. Then the property that the program always accesses x in a valid manner can be expressed by $(\langle \mathtt{close} \rangle \mathbf{true}) \land (\langle \mathtt{read} \rangle \langle \mathtt{close} \rangle \mathbf{true})$. Note that we have just replaced the non-deterministic branch with the logical conjunction, as we wish here to require that the program's behavior is valid in *both* branches. We can also deal with conditional branches if HFL is extended with predicates; $\mathbf{if}\ b\ \mathbf{then}\ \mathtt{close}(x)\ \mathbf{else}\ (\mathtt{read}(x); \mathtt{close}(x))$ can be translated to $(b \Rightarrow \langle \mathtt{close} \rangle \mathbf{true}) \land (\neg b \Rightarrow \langle \mathtt{read} \rangle \langle \mathtt{close} \rangle \mathbf{true})$. Let us also consider the recursive function f defined by:

$$f\, x = \mathtt{close}(x) \Box (\mathtt{read}(x); \mathtt{read}(x); f\, x),$$

Then, the program $f\, x$ being valid can be represented by using a (greatest) fixpoint formula:

$$\nu F.(\langle \mathtt{close} \rangle \mathbf{true}) \land (\langle \mathtt{read} \rangle \langle \mathtt{read} \rangle F).$$

If the state q_0 satisfies this formula (which is indeed the case), then we know that all the file accesses made by $f\, x$ are valid. So far, we have used only the modal μ-calculus formulas. If we wish to express the validity of higher-order programs, we need HFL formulas; such examples are given later.

[2] Here, for the sake of simplicity, we assume that we are interested in the usage of the single file pointer x, so that the name x can be ignored in HFL formulas; usage of multiple files can be tracked by using the technique of [18].

We generalize the above idea and formalize reductions from various classes of verification problems for simply-typed higher-order functional programs with recursion, integers and non-determinism – including verification of may/must-reachability, trace properties, and linear-time temporal properties (and their negations) – to (extended) HFL model checking where HFL is extended with integer predicates, and prove soundness and completeness of the reductions. Extended HFL model checking problems obtained by the reductions are (necessarily) undecidable in general, but for finite-data programs (i.e., programs that consist of only functions and data from finite data domains such as Booleans), the reductions yield *pure* HFL model checking problems, which are decidable [43].

Our reductions provide sound and complete logical characterizations of a wide range of program properties mentioned above. Nice properties of the logical characterizations include: (i) (like verification conditions for Hoare triples,) once the logical characterization is obtained as an HFL formula, purely logical reasoning can be used to prove or disprove it (without further referring to the program semantics); for that purpose, one may use theorem provers with various degrees of automation, ranging from interactive ones like Coq, semi-automated ones requiring some annotations, to fully automated ones (though the latter two are yet to be implemented), (ii) (unlike the standard verification condition generation for Hoare triples using invariant annotations) the logical characterization can *automatically* be computed, without any annotations, (iii) standard logical reasoning can be applied based on the semantics of formulas; for example, co-induction and induction can be used for proving ν- and μ-formulas respectively, and (iv) thanks to the completeness, the set of program properties characterizable by HFL formula is closed under negations; for example, from a formula characterizing may-reachability, one can obtain a formula characterizing non-reachability by just taking the De Morgan dual.

Compared with previous approaches based on HORS model checking [19, 23, 26, 33, 38], our approach based on (extended) HFL model checking provides more uniform, streamlined methods for higher-order program verification. HORS model checking provides sound and complete verification methods for *finite-data* programs [18, 19], but for infinite-data programs, other techniques such as predicate abstraction [23] and program transformation [27, 31] had to be combined to obtain sound (but incomplete) reductions to HORS model checking. Furthermore, the techniques were different for each of program properties, such as reachability [23], termination [27], non-termination [26], fair termination [31], and fair non-termination [44]. In contrast, our reductions are sound and complete even for infinite-data programs. Although the obtained HFL model checking problems are undecidable in general, the reductions allow us to treat various program properties uniformly; all the verifications are boiled down to the issue of how to prove μ- and ν-formulas (and as remarked above, we can use induction and co-induction to deal with them). Technically, our reduction to HFL model checking may actually be considered an extension of HORS model checking to infinite data programs, as discussed in Appendix H.

The rest of this paper is structured as follows. Section 2 introduces HFL extended with integer predicates and defines the HFL model checking problem. Section 3 informally demonstrates some examples of reductions from program verification problems to HFL model checking. Section 4 introduces a functional language used to formally discuss the reductions in later sections. Sections 5, 6, and 7 consider may/must-reachability, trace properties, and temporal properties respectively, and present (sound and complete) reductions from verification of those properties to HFL model checking. Section 8 discusses related work, and Section 9 concludes the paper. Proofs are found in Appendices.

2 (Extended) HFL

In this section, we introduce an extension of higher-order fixpoint logic (HFL) [43] with integer predicates (which we call $\mathrm{HFL}_{\mathbf{Z}}$; we often drop the subscript and just write HFL, as in Section 1), and define the $\mathrm{HFL}_{\mathbf{Z}}$ model checking problem. The set of integers can actually be replaced by another infinite set X of data (like the set of natural numbers or the set of finite trees) to yield HFL_X.

2.1 Syntax

For a map f, we write $dom(f)$ and $codom(f)$ for the domain and codomain of f respectively. We write $\mathbf{Z}$ for the set of integers, ranged over by the meta-variable n below. We assume a set **Pred** of primitive predicates on integers, ranged over by p. We write $\mathtt{arity}(p)$ for the arity of p. We assume that **Pred** contains standard integer predicates such as $=$ and $<$, and also assume that, for each predicate $p \in \mathbf{Pred}$, there also exists a predicate $\neg p \in \mathbf{Pred}$ such that, for any integers $n_1, \dots, n_k$, $p(n_1, \dots, n_k)$ holds if and only if $\neg p(n_1, \dots, n_k)$ does not hold; thus, $\neg p(n_1, \dots, n_k)$ should be parsed as $(\neg p)(n_1, \dots, n_k)$, but can semantically be interpreted as $\neg(p(n_1, \dots, n_k))$.

The syntax of *$\mathrm{HFL}_{\mathbf{Z}}$ formulas* is given by:

$$
\begin{aligned}
\varphi \text{ (formulas) } &::= n \mid \varphi_1 \ \mathtt{op} \ \varphi_2 \mid \mathbf{true} \mid \mathbf{false} \mid p(\varphi_1, \dots, \varphi_k) \mid X \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \\
&\quad \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X^{\tau}.\varphi \mid \nu X^{\tau}.\varphi \mid \lambda X : \sigma.\varphi \mid \varphi_1 \ \varphi_2 \\
\tau \text{ (types) } &::= \bullet \mid \sigma \to \tau \qquad \sigma \text{ (extended types) } ::= \tau \mid \mathtt{int}
\end{aligned}
$$

Here, $\mathtt{op}$ ranges over a set of binary operations on integers, such as $+$, and X ranges over a denumerable set of variables. We have extended the original HFL [43] with integer expressions (n and $\varphi_1 \ \mathtt{op} \ \varphi_2$), and atomic formulas $p(\varphi_1, \dots, \varphi_k)$ on integers (here, the arguments of integer operations or predicates will be restricted to integer expressions by the type system introduced below). Following [20], we have omitted negations, as any formula can be transformed to an equivalent negation-free formula [30].

We explain the meaning of each formula informally; the formal semantics is given in Section 2.2. Like modal μ-calculus [11, 25], each formula expresses a property of a labeled transition system. The first line of the syntax of formulas consists of the standard constructs of predicate logics. On the second line, as in the standard modal μ-calculus, $\langle a\rangle\varphi$ means that there exists an a-labeled transition to a state that satisfies φ. The formula $[a]\varphi$ means that after any a-labeled transition, φ is satisfied. The formulas $\mu X^{\tau}.\varphi$ and $\nu X^{\tau}.\varphi$ represent the least and greatest fixpoint formulas (the least and greatest formulas such that $X = \varphi$) respectively; unlike the modal μ-calculus, X may range over not only propositional variables but also higher-order predicate variables (of type τ). The λ-abstractions $\lambda X : \sigma.\varphi$ and applications $\varphi_1\ \varphi_2$ are used to manipulate higher-order predicates. We often omit type annotations in $\mu X^{\tau}.\varphi$, $\nu X^{\tau}.\varphi$ and $\lambda X : \sigma.\varphi$, and just write $\mu X.\varphi$, $\nu X.\varphi$ and $\lambda X.\varphi$.

Example 1. Consider the formula $\varphi_{\mathtt{ab}}\,\varphi$ where $\varphi_{\mathtt{ab}} = \mu X^{\bullet\to\bullet}.\lambda Y : \bullet.Y \vee \langle\mathtt{a}\rangle(X(\langle\mathtt{b}\rangle Y))$. We can expand the formula as follows:

$$\begin{aligned}\varphi_{\mathtt{ab}}\,\varphi &= (\lambda Y.\bullet.Y \vee \langle\mathtt{a}\rangle(\varphi_{\mathtt{ab}}(\langle\mathtt{b}\rangle Y)))\varphi = \varphi \vee \langle\mathtt{a}\rangle(\varphi_{\mathtt{ab}}(\langle\mathtt{b}\rangle\varphi))\\ &= \varphi \vee \langle\mathtt{a}\rangle(\langle\mathtt{b}\rangle\varphi \vee \langle\mathtt{a}\rangle(\varphi_{\mathtt{ab}}(\langle\mathtt{b}\rangle\langle\mathtt{b}\rangle\varphi))) = \cdots,\end{aligned}$$

and obtain $\varphi \vee (\langle\mathtt{a}\rangle\langle\mathtt{b}\rangle\varphi) \vee (\langle\mathtt{a}\rangle\langle\mathtt{a}\rangle\langle\mathtt{b}\rangle\langle\mathtt{b}\rangle\varphi) \vee \cdots$. Thus, the formula means that there is a transition sequence of the form $\mathtt{a}^n\mathtt{b}^n$ for some $n \geq 0$ that leads to a state satisfying φ.

Following [20], we exclude out unmeaningful formulas such as $(\langle a\rangle\mathbf{true}) + 1$ by using a simple type system.[3] The types $\bullet$, $\mathtt{int}$, and $\sigma \to \tau$ describe propositions, integers, and (monotonic) functions from σ to τ, respectively. Note that the integer type `int` may occur only in an argument position; this restriction is required to ensure that least and greatest fixpoints are well-defined. The typing rules for formulas are given in Figure 2. In the figure, Δ denotes a type environment, which is a finite map from variables to (extended) types. Below we consider only well-typed formulas, i.e., formulas φ such that $\Delta \vdash_{\mathtt{H}} \varphi : \tau$ for some Δ and τ.

2.2 Semantics and $\mathbf{HFL_Z}$ Model Checking

We now define the formal semantics of $\mathrm{HFL_Z}$ formulas. A *labeled transition system* (LTS) is a quadruple $\mathtt{L} = (U, A, \longrightarrow, \mathtt{s_{init}})$, where U is a finite set of states, A is a finite set of actions, $\longrightarrow\ \subseteq U \times A \times U$ is a labeled transition relation, and $\mathtt{s_{init}} \in U$ is the initial state. We write $\mathtt{s}_1 \xrightarrow{a} \mathtt{s}_2$ when $(\mathtt{s}_1, a, \mathtt{s}_2) \in \longrightarrow$.

For an LTS $\mathtt{L} = (U, A, \longrightarrow, \mathtt{s_{init}})$ and an extended type σ, we define the partially ordered set $(\mathcal{D}_{\mathtt{L},\sigma}, \sqsubseteq_{\mathtt{L},\sigma})$ inductively by:

$$\begin{aligned}&\mathcal{D}_{\mathtt{L},\bullet} = 2^U \qquad \sqsubseteq_{\mathtt{L},\bullet} = \subseteq \qquad \mathcal{D}_{\mathtt{L},\mathtt{int}} = \mathbf{Z} \qquad \sqsubseteq_{\mathtt{L},\mathtt{int}} = \{(n,n) \mid n \in \mathbf{Z}\}\\ &\mathcal{D}_{\mathtt{L},\sigma\to\tau} = \{f \in \mathcal{D}_{\mathtt{L},\sigma} \to \mathcal{D}_{\mathtt{L},\tau} \mid \forall x, y.(x \sqsubseteq_{\mathtt{L},\sigma} y \Rightarrow f\,x \sqsubseteq_{\mathtt{L},\tau} f\,y)\}\\ &\sqsubseteq_{\mathtt{L},\sigma\to\tau} = \{(f,g) \mid \forall x \in \mathcal{D}_{\mathtt{L},\sigma}.f(x) \sqsubseteq_{\mathtt{L},\tau} g(x)\}\end{aligned}$$

Note that $(\mathcal{D}_{\mathtt{L},\tau}, \sqsubseteq_{\mathtt{L},\tau})$ forms a complete lattice (but $(\mathcal{D}_{\mathtt{L},\mathtt{int}}, \sqsubseteq_{\mathtt{L},\mathtt{int}})$ does not). We write $\bot_{\mathtt{L},\tau}$ and $\top_{\mathtt{L},\tau}$ for the least and greatest elements of $\mathcal{D}_{\mathtt{L},\tau}$ (which are $\lambda\widetilde{x}.\emptyset$

[3] The original type system of [43] was more complex due to the presence of negations.

$$\frac{}{\Delta \vdash_{\mathtt{H}} n : \mathtt{int}} \quad \text{(HT-Int)}$$

$$\frac{\Delta \vdash_{\mathtt{H}} \varphi_i : \mathtt{int} \text{ for each } i \in \{0,1\}}{\Delta \vdash_{\mathtt{H}} \varphi_1 \,\mathtt{op}\, \varphi_2 : \mathtt{int}} \quad \text{(HT-Op)}$$

$$\frac{}{\Delta \vdash_{\mathtt{H}} \mathbf{true} : \bullet} \quad \text{(HT-True)}$$

$$\frac{}{\Delta \vdash_{\mathtt{H}} \mathbf{false} : \bullet} \quad \text{(HT-False)}$$

$$\frac{\mathtt{arity}(p) = k \qquad \Delta \vdash_{\mathtt{H}} \varphi_i : \mathtt{int} \text{ for each } i \in \{1,\dots,k\}}{\Delta \vdash_{\mathtt{H}} p(\varphi_1,\dots,\varphi_k) : \bullet} \quad \text{(HT-Pred)}$$

$$\frac{}{\Delta, X : \sigma \vdash_{\mathtt{H}} X : \sigma} \quad \text{(HT-Var)}$$

$$\frac{\Delta \vdash_{\mathtt{H}} \varphi_i : \bullet \text{ for each } i \in \{0,1\}}{\Delta \vdash_{\mathtt{H}} \varphi_1 \vee \varphi_2 : \bullet} \quad \text{(HT-Or)}$$

$$\frac{\Delta \vdash_{\mathtt{H}} \varphi_i : \bullet \text{ for each } i \in \{0,1\}}{\Delta \vdash_{\mathtt{H}} \varphi_1 \wedge \varphi_2 : \bullet} \quad \text{(HT-And)}$$

$$\frac{\Delta \vdash_{\mathtt{H}} \varphi : \bullet}{\Delta \vdash_{\mathtt{H}} \langle a \rangle \varphi : \bullet} \quad \text{(HT-Some)}$$

$$\frac{\Delta \vdash_{\mathtt{H}} \varphi : \bullet}{\Delta \vdash_{\mathtt{H}} [a] \varphi : \bullet} \quad \text{(HT-All)}$$

$$\frac{\Delta, X : \tau \vdash_{\mathtt{H}} \varphi : \tau}{\Delta \vdash_{\mathtt{H}} \mu X^{\tau}.\, \varphi : \tau} \quad \text{(HT-Mu)}$$

$$\frac{\Delta, X : \tau \vdash_{\mathtt{H}} \varphi : \tau}{\Delta \vdash_{\mathtt{H}} \nu X^{\tau}.\, \varphi : \tau} \quad \text{(HT-Nu)}$$

$$\frac{\Delta, X : \sigma \vdash_{\mathtt{H}} \varphi : \tau}{\Delta \vdash_{\mathtt{H}} \lambda X : \sigma.\, \varphi : \sigma \to \tau} \quad \text{(HT-Abs)}$$

$$\frac{\Delta \vdash_{\mathtt{H}} \varphi_1 : \sigma \to \tau \qquad \Delta \vdash_{\mathtt{H}} \varphi_2 : \sigma}{\Delta \vdash_{\mathtt{H}} \varphi_1\, \varphi_2 : \tau} \quad \text{(HT-App)}$$

Fig. 2. Typing Rules for $\mathrm{HFL}_{\mathbf{Z}}$ Formulas

and $\lambda\widetilde{x}.U$) respectively. We sometimes omit the subscript $\mathtt{L}$ below. Let $[\![\Delta]\!]_{\mathtt{L}}$ be the set of functions (called *valuations*) that maps X to an element of $\mathcal{D}_{\mathtt{L},\sigma}$ for each $X : \sigma \in \Delta$. For an HFL formula φ such that $\Delta \vdash_{\mathtt{H}} \varphi : \sigma$, we define $[\![\Delta \vdash_{\mathtt{H}} \varphi : \sigma]\!]_{\mathtt{L}}$ by induction on the derivation[4] of $\Delta \vdash_{\mathtt{H}} \varphi : \sigma$, as follows.

$$[\![\Delta \vdash_{\mathtt{H}} n : \mathtt{int}]\!]_{\mathtt{L}}(\rho) = n \qquad [\![\Delta \vdash_{\mathtt{H}} \mathbf{true} : \bullet]\!]_{\mathtt{L}}(\rho) = U \qquad [\![\Delta \vdash_{\mathtt{H}} \mathbf{false} : \bullet]\!]_{\mathtt{L}}(\rho) = \emptyset$$

$$[\![\Delta \vdash_{\mathtt{H}} \varphi_1 \,\mathtt{op}\, \varphi_2 : \mathtt{int}]\!]_{\mathtt{L}}(\rho) = ([\![\Delta \vdash_{\mathtt{H}} \varphi_1 : \mathtt{int}]\!]_{\mathtt{L}}(\rho))[\![\mathtt{op}]\!]([\![\Delta \vdash_{\mathtt{H}} \varphi_2 : \mathtt{int}]\!]_{\mathtt{L}}(\rho))$$

$$[\![\Delta \vdash_{\mathtt{H}} p(\varphi_1,\dots,\varphi_k) : \bullet]\!]_{\mathtt{L}}(\rho) = \begin{cases} U \text{ if } ([\![\Delta \vdash_{\mathtt{H}} \varphi_1 : \mathtt{int}]\!]_{\mathtt{L}}(\rho),\dots,[\![\Delta \vdash_{\mathtt{H}} \varphi_k : \mathtt{int}]\!]_{\mathtt{L}}(\rho)) \in [\![p]\!] \\ \emptyset \;\; \text{otherwise} \end{cases}$$

$$[\![\Delta, X : \sigma \vdash_{\mathtt{H}} X : \sigma]\!]_{\mathtt{L}}(\rho) = \rho(X)$$

$$[\![\Delta \vdash_{\mathtt{H}} \varphi_1 \vee \varphi_2 : \bullet]\!]_{\mathtt{L}}(\rho) = [\![\Delta \vdash_{\mathtt{H}} \varphi_1 : \bullet]\!]_{\mathtt{L}}(\rho) \cup [\![\Delta \vdash_{\mathtt{H}} \varphi_2 : \bullet]\!]_{\mathtt{L}}(\rho)$$

$$[\![\Delta \vdash_{\mathtt{H}} \varphi_1 \wedge \varphi_2 : \bullet]\!]_{\mathtt{L}}(\rho) = [\![\Delta \vdash_{\mathtt{H}} \varphi_1 : \bullet]\!]_{\mathtt{L}}(\rho) \cap [\![\Delta \vdash_{\mathtt{H}} \varphi_2 : \bullet]\!]_{\mathtt{L}}(\rho)$$

$$[\![\Delta \vdash_{\mathtt{H}} \langle a \rangle \varphi : \bullet]\!]_{\mathtt{L}}(\rho) = \{s \mid \exists s' \in [\![\Delta \vdash_{\mathtt{H}} \varphi : \bullet]\!]_{\mathtt{L}}(\rho).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\}$$

$$[\![\Delta \vdash_{\mathtt{H}} [a] \varphi : \bullet]\!]_{\mathtt{L}}(\rho) = \{\mathsf{s} \mid \forall \mathsf{s}' \in S.\ (\mathsf{s} \xrightarrow{a} \mathsf{s}' \text{ implies } \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi : \bullet]\!]_{\mathtt{L}}(\rho))\}$$

$$[\![\Delta \vdash_{\mathtt{H}} \mu X^{\tau}.\varphi : \tau]\!]_{\mathtt{L}}(\rho) = \mathbf{lfp}_{\mathtt{L},\tau}([\![\Delta \vdash_{\mathtt{H}} \lambda X : \tau.\ \varphi]\!]_{\mathtt{L}}(\rho))$$

$$[\![\Delta \vdash_{\mathtt{H}} \nu X^{\tau}.\varphi : \tau]\!]_{\mathtt{L}}(\rho) = \mathbf{gfp}_{\mathtt{L},\tau}([\![\Delta \vdash_{\mathtt{H}} \lambda X : \tau.\ \varphi]\!]_{\mathtt{L}}(\rho))$$

$$[\![\Delta \vdash_{\mathtt{H}} \lambda X : \sigma.\ \varphi : \sigma \to \tau]\!]_{\mathtt{L}}(\rho) = \{(v, [\![\Delta, X : \sigma \vdash_{\mathtt{H}} \varphi : \tau]\!]_{\mathtt{L}}(\rho[X \mapsto v])) \mid v \in \mathcal{D}_{\mathtt{L},\sigma}\}$$

$$[\![\Delta \vdash_{\mathtt{H}} \varphi_1\, \varphi_2 : \tau]\!]_{\mathtt{L}}(\rho) = [\![\Delta \vdash_{\mathtt{H}} \varphi_1 : \sigma \to \tau]\!]_{\mathtt{L}}(\rho)([\![\Delta \vdash_{\mathtt{H}} \varphi_2 : \sigma]\!]_{\mathtt{L}}(\rho))$$

Here, $[\![\mathtt{op}]\!]$ denotes the binary function on integers represented by $\mathtt{op}$ and $[\![p]\!]$ denotes the k-ary relation on integers represented by p. The least/greatest fixpoint

[4] Note that the derivation of each judgment $\Delta \vdash_{\mathtt{H}} \varphi : \sigma$ is unique if there is any.

operators $\mathbf{lfp}_\tau$ and $\mathbf{gfp}_\tau$ are defined by:

$$\mathbf{lfp}_{\mathtt{L},\tau}(f) = \bigsqcap_{\mathtt{L},\tau}\{x \in \mathcal{D}_{\mathtt{L},\tau} \mid f(x) \sqsubseteq_{\mathtt{L},\tau} x\} \qquad \mathbf{gfp}_{\mathtt{L},\tau}(f) = \bigsqcup_{\mathtt{L},\tau}\{x \in \mathcal{D}_{\mathtt{L},\tau} \mid x \sqsubseteq_{\mathtt{L},\tau} f(x)\}$$

Here, $\bigsqcup_{\mathtt{L},\tau}$ and $\bigsqcap_{\mathtt{L},\tau}$ denote respectively the least upper bound and the greatest lower bound with respect to $\sqsubseteq_{\mathtt{L},\tau}$.

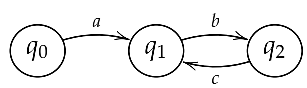
We often omit the subscript $\mathtt{L}$ and write $[\![\Delta \vdash_{\mathtt{H}} \varphi : \sigma]\!]$ for $[\![\Delta \vdash_{\mathtt{H}} \varphi : \sigma]\!]_{\mathtt{L}}$. For a closed formula, i.e., a formula well-typed under the empty type environment $\emptyset$, we often write $[\![\varphi]\!]_{\mathtt{L}}$ or just $[\![\varphi]\!]$ for $[\![\emptyset \vdash_{\mathtt{H}} \varphi : \sigma]\!]_{\mathtt{L}}(\emptyset)$.

Example 2. For the LTS $\mathtt{L}_{\mathit{file}}$ in Figure 1, we have:

$$\begin{array}{l}[\![\nu X^\bullet.(\langle\mathtt{close}\rangle\mathbf{true} \wedge \langle\mathtt{read}\rangle X)]\!] = \\ \quad \mathbf{gfp}_{\mathtt{L},\bullet}(\lambda x \in \mathcal{D}_{\mathtt{L},\bullet}.[\![X : \bullet \vdash \langle\mathtt{close}\rangle\mathbf{true} \wedge \langle\mathtt{read}\rangle X : \bullet]\!](\{X \mapsto x\})) = \{q_0\}.\end{array}$$

In fact, $x = \{q_0\} \in \mathcal{D}_{\mathtt{L},\bullet}$ satisfies the equation: $[\![X : \bullet \vdash \langle\mathtt{close}\rangle\mathbf{true} \wedge \langle\mathtt{read}\rangle X : \bullet]\!]_{\mathtt{L}}(\{X \mapsto x\}) = x$, and $x = \{q_0\} \in \mathcal{D}_{\mathtt{L},\bullet}$ is the greatest such element.

Consider the following LTS $\mathtt{L}_1$:



and $\varphi_{\mathtt{ab}}\,(\langle c\rangle\mathbf{true})$ where $\varphi_{\mathtt{ab}}$ is the one introduced in Example 1. Then, $[\![\varphi_{\mathtt{ab}}\,(\langle c\rangle\mathbf{true})]\!]_{\mathtt{L}_1} = \{q_0, q_2\}$.

Definition 1 (HFL$_{\mathbf{Z}}$ model checking). *For a closed formula φ of type $\bullet$, we write* $\mathtt{L}, \mathtt{s} \models \varphi$ *if* $\mathtt{s} \in [\![\varphi]\!]_{\mathtt{L}}$, *and write* $\mathtt{L} \models \varphi$ *if* $\mathtt{s}_{\mathtt{init}} \in [\![\varphi]\!]_{\mathtt{L}}$. HFL$_{\mathbf{Z}}$ model checking *is the problem of, given* $\mathtt{L}$ *and* φ, *deciding whether* $\mathtt{L} \models \varphi$ *holds.*

The HFL$_{\mathbf{Z}}$ model checking problem is *un*decidable, due to the presence of integers; in fact, the semantic domain $\mathcal{D}_{\mathtt{L},\sigma}$ is not finite for σ that contains $\mathtt{int}$. The undecidability is obtained as a corollary of the soundness and completeness of the reduction from the may-reachability problem to HFL model checking discussed in Section 5. For the fragment of pure HFL (i.e., HFL$_{\mathbf{Z}}$ without integers, which we write HFL$_\emptyset$ below), the model checking problem is decidable [43].

The *order* of an HFL$_{\mathbf{Z}}$ model checking problem $\mathtt{L} \stackrel{?}{\models} \varphi$ is the highest order of types of subformulas of φ, where the order of a type is defined by: $\mathtt{order}(\bullet) = \mathtt{order}(\mathtt{int}) = 0$ and $\mathtt{order}(\sigma \to \tau) = \max(\mathtt{order}(\sigma){+}1, \mathtt{order}(\tau))$. The complexity of order-k HFL$_\emptyset$ model checking is k-EXPTIME complete [1], but polynomial time in the size of HFL formulas under the assumption that the other parameters (the size of LTS and the largest size of types used in formulas) are fixed [20].

Remark 1. Though we do not have quantifiers on integers as primitives, we can encode them using fixpoint operators. Given a formula $\varphi : \mathtt{int} \to \bullet$, we can express $\exists x{:}\mathtt{int}.\varphi(x)$ and $\forall x{:}\mathtt{int}.\varphi(x)$ by $(\mu X^{\mathtt{int}\to\bullet}.\lambda x{:}\mathtt{int}.\varphi(x) \vee X(x{-}1) \vee X(x{+}1))0$ and $(\nu X^{\mathtt{int}\to\bullet}.\lambda x : \mathtt{int}.\varphi(x) \wedge X(x-1) \wedge X(x+1))0$ respectively.

2.3 HES

As in [20], we often write an $\mathrm{HFL}_{\mathbf{Z}}$ formula as a sequence of fixpoint equations, called a *hierarchical equation system* (HES).

Definition 2. *An (extended)* hierarchical equation system *(HES) is a pair* $(\mathcal{E}, \varphi)$ *where* $\mathcal{E}$ *is a sequence of fixpoint equations, of the form:*

$$X_1^{\tau_1} =_{\alpha_1} \varphi_1; \cdots ; X_n^{\tau_n} =_{\alpha_n} \varphi_n.$$

Here, $\alpha_i \in \{\mu, \nu\}$*. We assume that* $X_1 : \tau_1, \ldots, X_n : \tau_n \vdash_{\mathtt{H}} \varphi_i : \tau_i$ *holds for each* $i \in \{1, \ldots, n\}$*, and that* $\varphi_1, \ldots, \varphi_n, \varphi$ *do not contain any fixpoint operators.*

The HES $\Phi = (\mathcal{E}, \varphi)$ represents the $\mathrm{HFL}_{\mathbf{Z}}$ formula $\mathit{toHFL}(\mathcal{E}, \varphi)$ defined inductively by: $\mathit{toHFL}(\epsilon, \varphi) = \varphi$ and $\mathit{toHFL}(\mathcal{E}; X^\tau =_\alpha \varphi', \varphi) = \mathit{toHFL}([\alpha X^\tau.\varphi'/X]\mathcal{E}, [\alpha X^\tau.\varphi'/X]\varphi)$. Conversely, every $\mathrm{HFL}_{\mathbf{Z}}$ formula can be easily converted to an equivalent HES. In the rest of the paper, we often represent an $\mathrm{HFL}_{\mathbf{Z}}$ formula in the form of HES, and just call it an $\mathrm{HFL}_{\mathbf{Z}}$ formula. We write $[\![\Phi]\!]$ for $[\![\mathit{toHFL}(\Phi)]\!]$. An HES $(X_1^{\tau_1} =_{\alpha_1} \varphi_1; \cdots ; X_n^{\tau_n} =_{\alpha_n} \varphi_n, \varphi)$ can be normalized to $(X_0^{\tau_0} =_\nu \varphi; X_1^{\tau_1} =_{\alpha_1} \varphi_1; \cdots ; X_n^{\tau_n} =_{\alpha_n} \varphi_n, X_0)$ where τ_0 is the type of φ. Thus, we sometimes call just a sequence of equations $X_0^{\tau_0} =_\nu \varphi; X_1^{\tau_1} =_{\alpha_1} \varphi_1; \cdots ; X_n^{\tau_n} =_{\alpha_n} \varphi_n$ an HES, with the understanding that "the main formula" is the first variable X_0. Also, we often write $X^\tau\ x_1 \cdots x_k =_\alpha \varphi$ for the equation $X^\tau =_\alpha \lambda x_1.\cdots\lambda x_k.\varphi$. We often omit type annotations and just write $X =_\alpha \varphi$ for $X^\tau =_\alpha \varphi$.

Example 3. The formula $\varphi_2 = \varphi_{\mathtt{ab}}(\langle c\rangle\mathbf{true})$ in Example 2 is expressed as the following HES:

$$\left(X =_\mu \lambda Y : \bullet.Y \vee \langle \mathtt{a}\rangle(X(\langle \mathtt{b}\rangle Y)), \quad X(\langle c\rangle\mathbf{true})\right).$$

The formula $\nu X.\mu Y.\langle \mathtt{b}\rangle X \vee \langle \mathtt{a}\rangle Y$ (which means that the current state has a transition sequence of the form $(\mathtt{a}^*\mathtt{b})^\omega$) is expressed as the following HES:

$$\left((X =_\nu Y; Y =_\mu \langle \mathtt{b}\rangle X \vee \langle \mathtt{a}\rangle Y), \quad X\right).$$

3 Warming Up

To help readers get more familiar with $\mathrm{HFL}_{\mathbf{Z}}$ and the idea of reductions, we give here some variations of the examples of verification of file-accessing programs in Section 1, which are instances of the "resource usage verification problem" [16]. General reductions will be discussed in Sections 5–7, after the target language is set up in Section 4.

Consider the following OCaml-like program, which uses exceptions.

```
let readex x = read x; (if * then () else raise Eof) in
let rec f x = readex x; f x in
let d = open_in "foo" in try f d with Eof -> close d
```

Here, * represents a non-deterministic boolean value. The function `readex` reads the file pointer x, and then non-deterministically raises an end-of-file (`Eof`) exception. The main expression (on the third line) first opens file "foo", calls `f` to read the file repeatedly, and closes the file upon an end-of-file exception. Suppose, as in the example of Section 1, we wish to verify that the file "foo" is accessed following the protocol in Figure 1.

First, we can remove exceptions by representing an exception handler as a special continuation [6]:

```
let readex x h k = read x; (if * then k() else h()) in
let rec f x h k = readex x h (fun _ -> f x h k) in
let d = open_in "foo" in f d (fun _ -> close d) (fun _ -> ())
```

Here, we have added to each function two parameters `h` and `k`, which represent an exception handler and a (normal) continuation respectively.

Let Φ be $(\mathcal{E}, F\ \mathbf{true}\ (\lambda r.\langle\texttt{close}\rangle\mathbf{true})\ (\lambda r.\mathbf{true}))$ where $\mathcal{E}$ is:

$$\begin{aligned}&\mathit{Readex}\ x\ h\ k =_\nu \langle\texttt{read}\rangle(k\,\mathbf{true} \wedge h\,\mathbf{true});\\ &F\ x\ h\ k =_\nu \mathit{Readex}\ x\ h\ (\lambda r.F\ x\ h\ k).\end{aligned}$$

Here, we have just replaced read/close operations with the modal operators $\langle\texttt{read}\rangle$ and $\langle\texttt{close}\rangle$, non-deterministic choice with a logical conjunction, and the unit value ( ) with **true**. Then, $\mathsf{L}_{\mathit{file}} \models \Phi$ if and only if the program performs only valid accesses to the file (e.g., it does not access the file after a close operation), where $\mathsf{L}_{\mathit{file}}$ is the LTS shown in Figure 1. The correctness of the reduction can be informally understood by observing that there is a close correspondence between reductions of the program and those of the HFL formula above, and when the program reaches a read command `read` x, the corresponding formula is of the form $\langle\texttt{read}\rangle\cdots$, meaning that the read operation is valid in the current state; a similar condition holds also for close operations. We will present a formal proof of the general translation of this in Section 6.

Let us consider another example, which uses integers:

```
let rec f y x k = if y=0 then (close x; k())
                  else (read x; f (y-1) x k) in
let d = open_in "foo" in f n d (fun _ -> ())
```

Here, `n` is an integer constant. The function `f` reads `x` `y` times, and then calls the continuation `k`. Let $\mathsf{L}'_{\mathit{file}}$ be the LTS obtained by adding to $\mathsf{L}_{\mathit{file}}$ a new state q_2 and the transition $q_1 \xrightarrow{\mathsf{end}} q_2$ (which intuitively means that a program is allowed to terminate in the state q_1), and let Φ' be $(\mathcal{E}', F\ n\ \mathbf{true}\ (\lambda r.\langle\texttt{end}\rangle\mathbf{true}))$ where $\mathcal{E}'$ is:

$$F\ y\ x\ k =_\mu (y = 0 \Rightarrow \langle\texttt{close}\rangle(k\,\mathbf{true})) \wedge (y \neq 0 \Rightarrow \langle\texttt{read}\rangle(F\ (y-1)\ x\ k)).$$

Here, $p(\varphi_1, \ldots, \varphi_k) \Rightarrow \varphi$ is an abbreviation of $\neg p(\varphi_1, \ldots, \varphi_k) \vee \varphi$. Then, $\mathsf{L}'_{\mathit{file}} \models \Phi'$ if and only if (i) the program performs only valid accesses to the file, (ii) it eventually terminates, and (iii) the file is closed when the program terminates. Notice the use of μ instead of ν above; by using μ, we can express liveness properties. The property $\mathsf{L}'_{\mathit{file}} \models \Phi'$ indeed holds for $n \geq 0$, but not for $n < 0$. In fact, $F\ n\ x\ k$ is equivalent to **false** for $n < 0$, and $\langle\texttt{read}\rangle^n\langle\texttt{close}\rangle(k\ \mathbf{true})$ for $n \geq 0$.

4 Target Language

This section sets up, as the target of program verification, a call-by-name[5] higher-order functional language extended with events. The language is essentially the same as the one used by [44] for discussing fair non-termination.

4.1 Syntax and Typing

We assume a finite set **Ev** of names called *events*, ranged over by a, and a denumerable set of variables, ranged over by $x, y, \ldots$. Events are used to express temporal properties of programs. We write $\widetilde{x}$ ($\widetilde{t}$, resp.) for a sequence of variables (terms, resp.), and write $|\widetilde{x}|$ for the length of the sequence.

A *program* is a pair (D, t) consisting of a set D of function definitions $\{f_1\ \widetilde{x}_1 = t_1, \ldots, f_n\ \widetilde{x}_n = t_n\}$ and a term t. The set of terms, ranged over by t, is defined by:

$$t ::= (\,) \mid x \mid n \mid t_1 \ \mathtt{op}\ t_2 \mid \textbf{event}\ a; t \mid \textbf{if}\ p(t'_1, \ldots, t'_k)\ \textbf{then}\ t_1\ \textbf{else}\ t_2 \mid t_1 t_2 \mid t_1 \Box t_2.$$

Here, n and p range over the sets of integers and integer predicates as in HFL formulas. The expression **event** $a; t$ raises an event a, and then evaluates t. Events are used to encode program properties of interest. For example, an assertion **assert**(b) can be expressed as **if** b **then** ( ) **else** (**event** `fail`; Ω), where `fail` is an event that expresses an assertion failure and Ω is a non-terminating term. If program termination is of interest, one can insert "**event** end" to every program termination point and check whether an end event occurs. The expression $t_1 \Box t_2$ evaluates t_1 or t_2 in a non-deterministic manner; it can be used to model, e.g., unknown inputs from an environment. We use the meta-variable P for programs. When $P = (D, t)$ with $D = \{f_1\ \widetilde{x}_1 = t_1, \ldots, f_n\ \widetilde{x}_n = t_n\}$, we write **funs**($P$) for $\{f_1, \ldots, f_n\}$ (i.e., the set of function names defined in P). Using λ-abstractions, we sometimes write $f = \lambda\widetilde{x}.t$ for the function definition $f\ \widetilde{x} = t$. We also regard D as a map from function names to terms, and write $dom(D)$ for $\{f_1, \ldots, f_n\}$ and $D(f_i)$ for $\lambda\widetilde{x}_i.t_i$.

Any program (D, t) can be normalized to ($D \cup \{$**main** $= t\}$, **main**) where **main** is a name for the "main" function. We sometimes write just D for a program (D, **main**), with the understanding that D contains a definition of **main**.

We restrict the syntax of expressions using a type system. The set of *simple types*, ranged over by κ, is defined by:

$$\kappa ::= \star \mid \eta \to \kappa \qquad\qquad \eta ::= \kappa \mid \mathtt{int}.$$

The types $\star$, `int`, and $\eta \to \kappa$ describe the unit value, integers, and functions from η to κ respectively. Note that `int` is allowed to occur only in argument positions. We defer typing rules to Appendix A, as they are standard, except that we require that the righthand side of each function definition must have type $\star$; this restriction, as well as the restriction that `int` occurs only in argument positions, do not lose generality, as those conditions can be ensured by applying CPS transformation. We consider below only well-typed programs.

[5] Call-by-value programs can be handled by applying the CPS transformation before applying the reductions to HFL model checking.

4.2 Operational Semantics

We define the labeled transition relation $t \xrightarrow{\ell}_D t'$, where ℓ is either ϵ or an event name, as the least relation closed under the rules in Figure 3. We implicitly assume that the program (D, t) is well-typed, and this assumption is maintained throughout reductions by the standard type preservation property (which we omit to prove). In the rules for if-expressions, $[\![t'_i]\!]$ represents the integer value denoted by t'_i; note that the well-typedness of (D, t) guarantees that t'_i must be arithmetic expressions consisting of integers and integer operations; thus, $[\![t'_i]\!]$ is well defined. We often omit the subscript D when it is clear from the context. We write $t \xrightarrow{\ell_1\cdots\ell_k}{}^*_D t'$ if $t \xrightarrow{\ell_1}_D \cdots \xrightarrow{\ell_k}_D t'$. Here, ϵ is treated as an empty sequence; thus, for example, we write $t \xrightarrow{ab}{}^*_D t'$ if $t \xrightarrow{a}_D \xrightarrow{\epsilon}_D \xrightarrow{b}_D \xrightarrow{\epsilon}_D t'$.

$$\frac{}{\textbf{event } a; t \xrightarrow{a}_D t} \qquad \frac{f\,\widetilde{x} = u \in D \qquad |\widetilde{x}| = |\widetilde{t}|}{f\,\widetilde{t} \xrightarrow{\epsilon}_D [\widetilde{t}/\widetilde{x}]u} \qquad \frac{([\![t'_1]\!], \ldots, [\![t'_k]\!]) \in [\![p]\!]}{\textbf{if } p(t'_1, \ldots, t'_k) \textbf{ then } t_1 \textbf{ else } t_2 \xrightarrow{\epsilon}_D t_1}$$

$$\frac{i \in \{1, 2\}}{t_1 \Box t_2 \xrightarrow{\epsilon}_D t_i} \qquad \frac{([\![t'_1]\!], \ldots, [\![t'_k]\!]) \notin [\![p]\!]}{\textbf{if } p(t'_1, \ldots, t'_k) \textbf{ then } t_1 \textbf{ else } t_2 \xrightarrow{\epsilon}_D t_2}$$

Fig. 3. Labeled Transition Semantics

For a program $P = (D, t_0)$, we define the set $\textbf{Traces}(P)(\subseteq \textbf{Ev}^* \cup \textbf{Ev}^\omega)$ of *traces* by:

$$\begin{aligned}\textbf{Traces}(D, t_0) = \{&\ell_0 \cdots \ell_{n-1} \in (\{\epsilon\} \cup \textbf{Ev})^* \mid \forall i \in \{0, \ldots, n-1\}. t_i \xrightarrow{\ell_i}_D t_{i+1}\}\\ &\cup \{\ell_0\ell_1 \cdots \in (\{\epsilon\} \cup \textbf{Ev})^\omega \mid \forall i \in \omega. t_i \xrightarrow{\ell_i}_D t_{i+1}\}.\end{aligned}$$

Note that since the label ϵ is regarded as an empty sequence, $\ell_0\ell_1\ell_2 = aa$ if $\ell_0 = \ell_2 = a$ and $\ell_1 = \epsilon$, and an element of $(\{\epsilon\} \cup \textbf{Ev})^\omega$ is regarded as that of $\textbf{Ev}^* \cup \textbf{Ev}^\omega$. We write $\textbf{FinTraces}(P)$ and $\textbf{InfTraces}(P)$ for $\textbf{Traces}(P) \cap \textbf{Ev}^*$ and $\textbf{Traces}(P) \cap \textbf{Ev}^\omega$ respectively. The set of *full traces* $\textbf{FullTraces}(D, t_0)(\subseteq \textbf{Ev}^* \cup \textbf{Ev}^\omega)$ is defined as:

$$\begin{aligned}\{&\ell_0 \cdots \ell_{n-1} \in (\{\epsilon\} \cup \textbf{Ev})^* \mid t_n = (\,) \wedge \forall i \in \{0, \ldots, n-1\}. t_i \xrightarrow{\ell_i}_D t_{i+1}\}\\ &\cup \{\ell_0\ell_1 \cdots \in (\{\epsilon\} \cup \textbf{Ev})^\omega \mid \forall i \in \omega. t_i \xrightarrow{\ell_i}_D t_{i+1}\}.\end{aligned}$$

Example 4. The last example in Section 1 is modeled as $P_{\mathit{file}} = (D, f\,(\,))$, where $D = \{f\,x = (\textbf{event } \texttt{close}; (\,)) \Box (\textbf{event } \texttt{read}; \textbf{event } \texttt{read}; f\,x)\}$. We have:

$$\begin{aligned}&\textbf{Traces}(P) = \{\texttt{read}^n \mid n \geq 0\} \cup \{\texttt{read}^{2n}\texttt{close} \mid n \geq 0\} \cup \{\texttt{read}^\omega\}\\ &\textbf{FinTraces}(P) = \{\texttt{read}^n \mid n \geq 0\} \cup \{\texttt{read}^{2n}\texttt{close} \mid n \geq 0\}\\ &\textbf{InfTraces}(P) = \{\texttt{read}^\omega\} \qquad \textbf{FullTraces}(P) = \{\texttt{read}^{2n}\texttt{close} \mid n \geq 0\} \cup \{\texttt{read}^\omega\}.\end{aligned}$$

5 May/Must-Reachability Verification

Here we consider the following problems:

- May-reachability: "Given a program P and an event a, may P raise a?"
- Must-reachability: "Given a program P and an event a, must P raise a?"

Since we are interested in a particular event a, we restrict here the event set **Ev** to a singleton set of the form $\{a\}$. Then, the may-reachability is formalized as $a \overset{?}{\in} \mathbf{Traces}(P)$, whereas the must-reachability is formalized as "does every trace in $\mathbf{FullTraces}(P)$ contain a?" We encode both problems into the validity of $\mathrm{HFL}_{\mathbf{Z}}$ formulas (without any modal operators $\langle a \rangle$ or $[a]$), or the $\mathrm{HFL}_{\mathbf{Z}}$ model checking of those formulas against a trivial model (which consists of a single state without any transitions). Since our reductions are sound and complete, the characterizations of their negations –non-reachability and may-non-reachability– can also be obtained immediately. Although these are the simplest classes of properties among those discussed in Sections 5–7, they are already large enough to accommodate many program properties discussed in the literature, including lack of assertion failures/uncaught exceptions [23] (which can be characterized as non-reachability; recall the encoding of assertions in Section 4), termination [27, 29] (characterized as must-reachability), and non-termination [26] (characterized as may-non-reachability).

5.1 May-Reachability

As in the examples in Section 3, we translate a program to a formula that says "the program may raise an event a" in a compositional manner. For example, **event** $a; t$ can be translated to **true** (since the event will surely be raised immediately), and $t_1 \Box t_2$ can be translated to $t_1^\dagger \vee t_2^\dagger$ where $t_i^\dagger$ is the result of the translation of t_i (since only one of t_1 and t_2 needs to raise an event).

Definition 3. *Let $P = (D, t)$ be a program. $\Phi_{P,may}$ is the HES $(D^{\dagger_{may}}, t^{\dagger_{may}})$, where $D^{\dagger_{may}}$ and $t^{\dagger_{may}}$ are defined by:*

$$\{f_1\,\widetilde{x_1} = t_1, \dots, f_n\,\widetilde{x_n} = t_n\}^{\dagger_{may}} = \left(f_1\,\widetilde{x_1} =_\mu {t_1}^{\dagger_{may}}; \cdots ; f_n\,\widetilde{x_n} =_\mu {t_n}^{\dagger_{may}}\right)$$
$$(\,)^{\dagger_{may}} = \mathbf{false} \qquad x^{\dagger_{may}} = x \qquad n^{\dagger_{may}} = n \qquad (t_1 \;\mathsf{op}\; t_2)^{\dagger_{may}} = {t_1}^{\dagger_{may}} \;\mathsf{op}\; {t_2}^{\dagger_{may}}$$
$$(\mathbf{if}\; p(t'_1, \dots, t'_k)\;\mathbf{then}\; t_1 \;\mathbf{else}\; t_2)^{\dagger_{may}} =$$
$$(p({t'_1}^{\dagger_{may}}, \dots, {t'_k}^{\dagger_{may}}) \wedge {t_1}^{\dagger_{may}}) \vee (\neg p({t'_1}^{\dagger_{may}}, \dots, {t'_k}^{\dagger_{may}}) \wedge {t_2}^{\dagger_{may}})$$
$$(\mathbf{event}\; a; t)^{\dagger_{may}} = \mathbf{true} \qquad (t_1 t_2)^{\dagger_{may}} = {t_1}^{\dagger_{may}} {t_2}^{\dagger_{may}} \qquad (t_1 \Box t_2)^{\dagger_{may}} = {t_1}^{\dagger_{may}} \vee {t_2}^{\dagger_{may}}.$$

Note that, in the definition of $D^{\dagger_{may}}$, the order of function definitions in D does not matter (i.e., the resulting HES is unique up to the semantic equality), since all the fixpoint variables are bound by μ.

Example 5. Consider the program $P_{loop} = (\{loop\; x = loop\; x\}, loop(\mathbf{event}\; a; (\,)))$. It is translated to the HES $\Phi_{loop} = (loop\; x =_\mu loop\; x, loop(\mathbf{true}))$. Since $loop \equiv \mu loop.\lambda x.loop\; x$ is equivalent to $\lambda x.\mathbf{false}$, Φ_{loop} is equivalent to **false**. In fact, P_{loop} never raises an event a (recall that our language is call-by-name).

Example 6. Consider the program $P_{sum} = (D_{sum}, \textbf{main})$ where D_{sum} is:

$$\begin{array}{l}\textbf{main} = sum\ n\ (\lambda r.\textbf{assert}(r \geq n))\\ sum\ x\ k = \textbf{if}\ x = 0\ \textbf{then}\ k\,0\ \textbf{else}\ sum\ (x-1)\ (\lambda r.k(x+r))\end{array}$$

Here, n is some integer constant, and $\textbf{assert}(b)$ is the macro introduced in Section 4. We have used λ-abstractions for the sake of readability. The function sum is a CPS version of a function that computes the summation of integers from 1 to x. The main function computes the sum $r = 1 + \cdots + n$, and asserts $r \geq n$. It is translated to the HES $\Phi_{P_2,path} = (\mathcal{E}_{sum}, \textbf{main})$ where $\mathcal{E}_{sum}$ is:

$$\begin{array}{l}\textbf{main} =_\mu sum\ n\ (\lambda r.(r \geq n \wedge \textbf{false}) \vee (r < n \wedge \textbf{true}))\\ sum\ x\ k =_\mu (x = 0 \wedge k\,0) \vee (x \neq 0 \wedge sum\ (x-1)\ (\lambda r.k(x+r))).\end{array}$$

Here, n is treated as a constant. Since the shape of the formula does not depend on the value of n, however, the property "an assertion failure may occur for some n" can be expressed by $\exists n.\Phi_{P_2,path}$.

Thanks to the completeness of the encoding (Theorem 1 below), the lack of assertion failures can be characterized by $\forall n.\Phi$, where Φ is the De Morgan dual of the above HES:

$$\begin{array}{l}\textbf{main} =_\nu sum\ n\ (\lambda r.(r < n \vee \textbf{true}) \wedge (r \geq n \vee \textbf{false}))\\ sum\ x\ k =_\nu (x \neq 0 \vee k\,0) \wedge (x = 0 \vee sum\ (x-1)\ (\lambda r.k(x+r))).\end{array}$$

□

The following theorem states that $\Phi_{P,may}$ is a complete characterization of the may-reachability of P.

Theorem 1. *Let P be a program. Then, $a \in \textbf{Traces}(P)$ if and only if $\mathtt{L}_0 \models \Phi_{P,may}$ for $\mathtt{L}_0 = (\{\mathtt{s}_\star\}, \emptyset, \emptyset, \mathtt{s}_\star)$.*

To prove the theorem, we first show the theorem for recursion-free programs and then lift it to arbitrary programs by using the continuity of functions represented in the fixpoint-free fragment of $\mathrm{HFL}_\mathbf{Z}$ formulas. See Appendix B.1 for a concrete proof.

5.2 Must-Reachability

The characterization of must-reachability can be obtained by an easy modification of the characterization of may-reachability: we just need to replace branches with logical conjunction.

Definition 4. *Let $P = (D, t)$ be a program. $\Phi_{P,must}$ is the HES $(D^{\dagger_{must}}, t^{\dagger_{must}})$, where $D^{\dagger_{must}}$ and $t^{\dagger_{must}}$ are defined by:*

$$\begin{array}{l}\{f_1\,\widetilde{x}_1 = t_1, \dots, f_n\,\widetilde{x}_n = t_n\}^{\dagger_{must}} = f_1\,\widetilde{x}_1 =_\mu {t_1}^{\dagger_{must}}; \cdots ; f_n\,\widetilde{x}_n =_\mu {t_n}^{\dagger_{must}}\\ (\,)^{\dagger_{must}} = \textbf{false} \qquad x^{\dagger_{must}} = x \qquad n^{\dagger_{must}} = n \qquad (t_1\ \mathtt{op}\ t_2)^{\dagger_{must}} = {t_1}^{\dagger_{must}}\ \mathtt{op}\ {t_2}^{\dagger_{must}}\\ (\textbf{if}\ p(t'_1, \dots, t'_k)\ \textbf{then}\ t_1\ \textbf{else}\ t_2)^{\dagger_{must}} =\\ \qquad (p({t'_1}^{\dagger_{must}}, \dots, {t'_k}^{\dagger_{must}}) \Rightarrow {t_1}^{\dagger_{must}}) \wedge (\neg p({t'_1}^{\dagger_{must}}, \dots, {t'_k}^{\dagger_{must}}) \Rightarrow {t_2}^{\dagger_{must}})\\ (\textbf{event}\ a; t)^{\dagger_{must}} = \textbf{true} \qquad (t_1 t_2)^{\dagger_{must}} = {t_1}^{\dagger_{must}}{t_2}^{\dagger_{must}} \qquad (t_1 \Box t_2)^{\dagger_{must}} = {t_1}^{\dagger_{must}} \wedge {t_2}^{\dagger_{must}}.\end{array}$$

Here, $p(\varphi_1, \dots, \varphi_k) \Rightarrow \varphi$ is a shorthand for $\neg p(\varphi_1, \dots, \varphi_k) \vee \varphi$.

We write $\mathbf{Must}_a(P)$ if every $\pi \in \mathbf{FullTraces}(P)$ contains a.

Example 7. Consider $P_{\texttt{loop}} = (D, \texttt{loop}\, m\, n)$ where D is:

$$\begin{aligned}\texttt{loop}\ x\ y = \mathbf{if}\ x \le 0 \lor y \le 0\ \mathbf{then}\ (\mathbf{event}\ \mathsf{end}; ())\\ \mathbf{else}\ (\texttt{loop}\ (x-1)\ (y * y)) \Box (\texttt{loop}\ x\ (y-1))\end{aligned}$$

Here, the event end is used to signal the termination of the program. The function `loop` non-deterministically updates the values of x and y until either x or y becomes non-positive. The must-termination of the program is characterized by $\Phi_{P_{\texttt{loop}},must} = (\mathcal{E}, \texttt{loop}\, m\, n)$ where $\mathcal{E}$ is:

$$\begin{aligned}\texttt{loop}\ x\ y =_\mu\ & (x \le 0 \lor y \le 0 \Rightarrow \mathbf{true})\\ & \land(\neg(x \le 0 \lor y \le 0) \Rightarrow (\texttt{loop}\ (x-1)\ (y * y)) \land (\texttt{loop}\ x\ (y-1))).\end{aligned}$$

The following theorem guarantees that $\Phi_{P,must}$ is indeed a sound and complete characterization of the must-reachability.

Theorem 2. *Let P be a program. Then, $\mathbf{Must}_a(P)$ if and only if $\mathtt{L}_0 \models \Phi_{P,must}$ for $\mathtt{L}_0 = (\{\mathtt{s}_\star\}, \emptyset, \emptyset, \mathtt{s}_\star)$.*

The proof is similar to that of Theorem 1; see Appendix B.2.

6 Trace Properties

Here we consider the verification problem: "Given a (non-ω) regular language L and a program P, does *every* finite event sequence of P belong to L? (i.e. $\mathbf{FinTraces}(P) \overset{?}{\subseteq} L$)" and reduce it to an $\mathrm{HFL}_{\mathbf{Z}}$ model checking problem. The verification of file-accessing programs considered in Section 3 may be considered an instance of the problem.[6]

Here we assume that the language L is closed under the prefix operation; this does not lose generality because $\mathbf{FinTraces}(P)$ is also closed under the prefix operation. We write $A_L = (Q, \Sigma, \delta, q_0)$ for the minimal, deterministic automaton with no dead states (hence the transition function δ may be partial). Since L is prefix-closed, $w \in L$ if and only if $\hat{\delta}(q_0, w)$ is defined (where $\hat{\delta}$ is defined by: $\hat{\delta}(q, \epsilon) = q$ and $\hat{\delta}(q, aw) = \hat{\delta}(\delta(q,a), w)$). We use the corresponding LTS $\mathtt{L}_L = (Q, \Sigma, \{(q, a, q') \mid \delta(q,a) = q'\}, q_0)$ as the model of the reduced $\mathrm{HFL}_{\mathbf{Z}}$ model checking problem.

Given the LTS $\mathtt{L}_L$ above, whether an event sequence $a_1 \cdots a_k$ belongs to L can be expressed as $\mathtt{L}_L \overset{?}{\models} \langle a_1 \rangle \cdots \langle a_k \rangle \mathbf{true}$. Whether all the event sequences in $\{a_{j,1} \cdots a_{j,k_j} \mid j \in \{1, \dots, n\}\}$ belong to L can be expressed as $\mathtt{L}_L \overset{?}{\models} \bigwedge_{j \in \{1,\dots,n\}} \langle a_{j,1} \rangle \cdots \langle a_{j,k_j} \rangle \mathbf{true}$. We can lift these translations for event sequences to the translation from a program (which can be considered a description of a set of event sequences) to an $\mathrm{HFL}_{\mathbf{Z}}$ formula, as follows.

[6] The last example in Section 3 is actually a combination with the must-reachability problem.

Definition 5. *Let $P = (D, t)$ be a program. $\Phi_{P,path}$ is the HES $(D^{\dagger_{path}}, t^{\dagger_{path}})$, where $D^{\dagger_{path}}$ and $t^{\dagger_{path}}$ are defined by:*

$$\{f_1\,\widetilde{x}_1 = t_1, \ldots, f_n\,\widetilde{x}_n = t_n\}^{\dagger_{path}} = f_1\,\widetilde{x}_1 =_\nu t_1{}^{\dagger_{path}}; \cdots ; f_n\,\widetilde{x}_n =_\nu t_n{}^{\dagger_{path}}$$
$$(\,)^{\dagger_{path}} = \mathbf{true} \qquad x^{\dagger_{path}} = x \qquad n^{\dagger_{path}} = n \qquad (t_1\ \mathtt{op}\ t_2)^{\dagger_{path}} = t_1{}^{\dagger_{path}}\ \mathtt{op}\ t_2{}^{\dagger_{path}}$$
$$(\mathbf{if}\ p(t'_1, \ldots, t'_k)\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2)^{\dagger_{path}} =$$
$$(p(t'_1{}^{\dagger_{path}}, \ldots, t'_k{}^{\dagger_{path}}) \Rightarrow t_1{}^{\dagger_{path}}) \wedge (\neg p(t'_1{}^{\dagger_{path}}, \ldots, t'_k{}^{\dagger_{path}}) \Rightarrow t_2{}^{\dagger_{path}})$$
$$(\mathbf{event}\ a; t)^{\dagger_{path}} = \langle a \rangle t^{\dagger_{path}} \qquad (t_1 t_2)^{\dagger_{path}} = t_1{}^{\dagger_{path}} t_2{}^{\dagger_{path}} \qquad (t_1 \Box t_2)^{\dagger_{path}} = t_1{}^{\dagger_{path}} \wedge t_2{}^{\dagger_{path}}.$$

Example 8. The last program discussed in Section 3 is modeled as $P_2 = (D_2, f\ m\ g)$, where m is an integer constant and D_2 consists of:

$$f\ n\ k = \mathbf{if}\ n = 0\ \mathbf{then}\ (\mathbf{event}\ \mathtt{close}; k\,(\,))\ \mathbf{else}\ (\mathbf{event}\ \mathtt{read}; f\ (n-1)\ k)$$
$$g\ r = \mathbf{event}\ \mathtt{end}; (\,)$$

Here, we have modeled accesses to the file, and termination as events. Then, $\Phi_{P_2,path} = (\mathcal{E}_{P_2,path}, f\ m\ g)$ where $\mathcal{E}_{P_2,path}$ is:[7]

$$f\ n\ k =_\nu (n = 0 \Rightarrow \langle \mathtt{close} \rangle (k\,\mathbf{true})) \wedge (n \neq 0 \Rightarrow \langle \mathtt{read} \rangle (f\ (n-1)\ k))$$
$$g\ r =_\nu \langle \mathtt{end} \rangle \mathbf{true}.$$

Let L be the prefix-closure of $\mathtt{read}^* \cdot \mathtt{close} \cdot \mathtt{end}$. Then $\mathtt{L}_L$ is $\mathtt{L}'_{file}$ in Section 3, and **FinTraces**$(P_2) \subseteq L$ can be verified by checking $\mathtt{L}_L \models \Phi_{P_2,path}$. □

Theorem 3. *Let P be a program and L be a regular, prefix-closed language. Then,* **FinTraces**$(P) \subseteq L$ *if and only if* $\mathtt{L}_L \models \Phi_{P,path}$.

As in Section 5, we first prove the theorem for programs in normal form, and then lift it to recursion-free programs by using the preservation of the semantics of $\text{HFL}_\mathbf{Z}$ formulas by reductions, and further to arbitrary programs by using the (co-)continuity of the functions represented by fixpoint-free $\text{HFL}_\mathbf{Z}$ formulas. The proof is given in Appendix C.

7 Linear-Time Temporal Properties

This section considers the following problem: "Given a program P and an ω-regular word language L, does **InfTraces**$(P) \cap L = \emptyset$ hold?" From the viewpoint of program verification, L represents the set of "bad" behaviors. This can be considered an extension of the problems considered in the previous sections.[8]

The reduction to HFL model checking is more involved than those in the previous sections. To see the difficulty, consider the program P_0:

$$(\{f = \mathbf{if}\ c\ \mathbf{then}\ (\mathbf{event}\ \mathtt{a}; f)\ \mathbf{else}\ (\mathbf{event}\ \mathtt{b}; f)\}, \quad f),$$

[7] Unlike in Section 3, the variables are bound by ν since we are not concerned with the termination property here.

[8] Note that finite traces can be turned into infinite ones by inserting a dummy event for every function call and replacing each occurrence of the unit value $(\,)$ with $loop(\,)$ where $loop\ x = \mathbf{event}\ \mathtt{dummy}; loop\ x$.

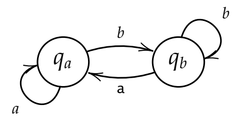


Fig. 4. LTS for $(a^*b)^\omega$

where c is some boolean expression. Let L be the complement of $(\mathtt{a}^*\mathtt{b})^\omega$, i.e., the set of infinite sequences that contain only finitely many b's. Following Section 6 (and noting that $\mathbf{InfTraces}(P) \cap L = \emptyset$ is equivalent to $\mathbf{InfTraces}(P) \subseteq (\mathtt{a}^*\mathtt{b})^\omega$ in this case), one may be tempted to prepare an LTS like the one in Figure 4 (which corresponds to the transition function of a (parity) word automaton accepting $(\mathtt{a}^*\mathtt{b})^\omega$), and translate the program to an HES Φ_{P_0} of the form:

$$(f =_\alpha (c \Rightarrow \langle \mathtt{a} \rangle f) \wedge (\neg c \Rightarrow \langle \mathtt{b} \rangle f), \quad f),$$

where α is μ or ν. However, such a translation would not work. If $c = \mathbf{true}$, then $\mathbf{InfTraces}(P_0) = \mathtt{a}^\omega$, hence $\mathbf{InfTraces}(P_0) \cap L \neq \emptyset$; thus, α should be μ for Φ_{P_0} to be unsatisfied. If $c = \mathbf{false}$, however, $\mathbf{InfTraces}(P_0) = \mathtt{b}^\omega$, hence $\mathbf{InfTraces}(P_0) \cap L = \emptyset$; thus, α must be ν for Φ_{P_0} to be satisfied.

The example above suggests that we actually need to distinguish between the two occurrences of f in the body of f's definition. Note that in the then- and else-clauses respectively, f is called after different events a and b. This difference is important, since we are interested in whether b occurs infinitely often. We thus duplicate f, and replace the program with the following program P_{dup}:

$$\begin{aligned}(\{&f_b = \mathbf{if}\ c\ \mathbf{then}\ (\mathbf{event}\ \mathtt{a}; f_a)\ \mathbf{else}\ (\mathbf{event}\ \mathtt{b}; f_b),\\ &f_a = \mathbf{if}\ c\ \mathbf{then}\ (\mathbf{event}\ \mathtt{a}; f_a)\ \mathbf{else}\ (\mathbf{event}\ \mathtt{b}; f_b)\}, f_b).\end{aligned}$$

For checking $\mathbf{InfTraces}(P_0) \cap L = \emptyset$, it is now sufficient to check that f_b is recursively called infinitely often. We can thus obtain the following HES:

$$\begin{aligned}((&f_b =_\nu (c \Rightarrow \langle \mathtt{a} \rangle f_a) \wedge (\neg c \Rightarrow \langle \mathtt{b} \rangle f_b);\\ &f_a =_\mu (c \Rightarrow \langle \mathtt{a} \rangle f_a) \wedge (\neg c \Rightarrow \langle \mathtt{b} \rangle f_b)), f_b).\end{aligned}$$

Note that f_b and f_a are bound by ν and μ respectively, reflecting the fact that b should occur infinitely often, but a need not. If $c = \mathbf{true}$, the formula is equivalent to $\nu f_b.\langle \mathtt{a} \rangle \mu f_a.\langle \mathtt{a} \rangle f_a$, which is false. If $c = \mathbf{false}$, then the formula is equivalent to $\nu f_b.\langle \mathtt{b} \rangle f_b$, which is satisfied by by the LTS in Figure 4.

The general translation is more involved due to the presence of higher-order functions, but, as in the example above, the overall translation consists of two steps. We first replicate functions according to what events may occur between two recursive calls, and reduce the problem $\mathbf{InfTraces}(P) \cap L \stackrel{?}{=} \emptyset$ to a problem of analyzing which functions are recursively called infinitely often, which we call a *call-sequence analysis*. We can then reduce the call-sequence analysis to HFL model checking in a rather straightforward manner.[9] The resulting HFL formula actually does not contain modal operators.[10] So, as in Section 5, the resulting problem is the validity checking of HFL formulas without modal operators.

In the rest of this section, we first introduce the call-sequence analysis problem and its reduction to HFL model checking in Section 7.1. We then show how to reduce the temporal verification problem $\mathbf{InfTraces}(P) \cap L \stackrel{?}{=} \emptyset$ to an instance of the call-sequence analysis problem in Section 7.2.

7.1 Call-sequence analysis

We define the call-sequence analysis and reduce it to an HFL model-checking problem. As mentioned above, in the call-sequence analysis, we are interested in analyzing which functions are *recursively called* infinitely often. Here, we say that g is *recursively called from* f, if $f\,\widetilde{s} \xrightarrow{\epsilon}_D [\widetilde{s}/\widetilde{x}]t_f \xrightarrow{\widetilde{\ell}}{}^*_D g\,\widetilde{t}$, where $f\,\widetilde{x} = t_f \in D$ and g "originates from" t_f (a more formal definition will be given in Definition 6 below). For example, consider the following program P_{app}, which is a twisted version of P_{dup} above.

$$
\begin{aligned}
(\{&\mathsf{app}\,h\,x = h\,x,\\
&f_b\,x = \mathbf{if}\ x > 0\ \mathbf{then}\ (\mathbf{event}\ \mathsf{a};\mathsf{app}\,f_a\,(x-1))\ \mathbf{else}\ (\mathbf{event}\ \mathsf{b};\mathsf{app}\,f_b\,5),\\
&f_a\,x = \mathbf{if}\ x > 0\ \mathbf{then}\ (\mathbf{event}\ \mathsf{a};\mathsf{app}\,f_a\,(x-1))\ \mathbf{else}\ (\mathbf{event}\ \mathsf{b};\mathsf{app}\,f_b\,5)\},\ f_b\,5).
\end{aligned}
$$

Then f_a is "recursively called" from f_b in $f_b\,5 \xrightarrow{\mathsf{a}}{}^*_D \mathsf{app}\,f_a\,4 \xrightarrow{\epsilon}{}^*_D f_a\,4$ (and so is app). We are interested in infinite chains of recursive calls $f_0 f_1 f_2 \cdots$, and which functions may occur infinitely often in each chain. For instance, the program above has the unique infinite chain $(f_b f_a^5)^\omega$, in which both f_a and f_b occur infinitely often. (Besides the infinite chain, the program has finite chains like $f_b\,\mathsf{app}$; note that the chain cannot be extended further, as the body of app does not have any occurrence of recursive functions: app, f_a and f_b.)

We define the notion of "recursive calls" and call-sequences more formally below.

Definition 6 (recursive call relation, call sequences). *Let $P = (D, f_1\,\widetilde{s})$ be a program, with $D = \{f_i\,\tilde{x}_i = u_i\}_{1\le i\le n}$. We define $D^\sharp := D \cup \{f_i^\sharp\,\tilde{x} = u_i\}_{1\le i\le n}$ where $f_1^\sharp, \dots, f_n^\sharp$ are fresh symbols. Thus, $D^\sharp$ has two copies of each function symbol, one of which is marked by $\sharp$. For the terms $\widetilde{t_i}$ and $\widetilde{t_j}$ that do not contain marked symbols, we write $f_i\,\widetilde{t_i} \rightsquigarrow_D f_j\,\widetilde{t_j}$ if (i) $[\widetilde{t_i}/\widetilde{x_i}][f_1^\sharp/f_1, \dots, f_n^\sharp/f_n]u_i \xrightarrow{\widetilde{\ell}}{}^*_{D^\sharp} f_j^\sharp\,\widetilde{t'_j}$ and (ii) $\widetilde{t_j}$ is obtained by erasing all the marks in $\widetilde{t'_j}$. We write* $\mathbf{Callseq}(P)$ *for the set of (possibly infinite) sequences of function symbols:*

$$\{f_1\,g_1\,g_2 \cdots \mid f_1\,\widetilde{s} \rightsquigarrow_D g_1\,\widetilde{t_1} \rightsquigarrow_D g_2\,\widetilde{t_2} \rightsquigarrow_D \cdots\}.$$

[9] The proof of the correctness is however non-trivial.

[10] In the example above, we can actually remove ⟨a⟩ and ⟨b⟩, as information about events has been taken into account when f was duplicated.

We write **InfCallseq**(P) *for the subset of* **Callseq**(P) *consisting of infinite sequences, i.e.,* **Callseq**$(P) \cap \{f_1, \ldots, f_n\}^\omega$.

For example, for P_{app} above, **Callseq**(P) is the prefix closure of $\{(f_b f_a^5)^\omega\} \cup \{s, s \cdot \mathtt{app} \mid s \text{ is a finite prefix of } (f_b f_a^5)^\omega\}$, and **InfCallseq**$(P)$ is the singleton set $\{(f_b f_a^5)^\omega\}$.

Definition 7 (Call-sequence analysis). *A* priority assignment *for a program P is a function $\Omega : \mathbf{funs}(P) \to \mathbb{N}$ from the set of function symbols of P to the set $\mathbb{N}$ of natural numbers. We write $\models_{csa} (P, \Omega)$ if every infinite call-sequence $g_0 g_1 g_2 \cdots \in$* **InfCallseq**(P) *satisfies the parity condition w.r.t. Ω, i.e., the largest number occurring infinitely often in $\Omega(g_0)\Omega(g_1)\Omega(g_2)\ldots$ is even.* Call-sequence analysis *is the problem of, given a program P with a priority assignment Ω, deciding whether $\models_{csa} (P, \Omega)$ holds.*

For example, for P_{app} and the priority assignment $\Omega_{app} = \{\mathtt{app} \mapsto 3, f_a \mapsto 1, f_b \mapsto 2\}$, $\models_{csa} (P_{app}, \Omega_{app})$ holds.

The call-sequence analysis can naturally be reduced to the HFL model checking against the trivial LTS $\mathtt{L}_0 = (\{\mathtt{s}_\star\}, \emptyset, \emptyset, \mathtt{s}_\star)$ (or the validity checking).

Definition 8. *Let $P = (D, t)$ be a program and Ω be a priority assignment for P. The HES $\Phi_{(P,\Omega),csa}$ is $(D^{\dagger_{csa}}, t^{\dagger_{csa}})$, where $D^{\dagger_{csa}}$ and $t^{\dagger_{csa}}$ are defined by:*

$$\{f_1\,\widetilde{x}_1 = t_1, \ldots, f_n\,\widetilde{x}_n = t_n\}^{\dagger_{csa}} = \left(f_1\,\widetilde{x}_1 =_{\alpha_1} t_1^{\dagger_{csa}}; \cdots; f_n\,\widetilde{x}_n =_{\alpha_n} t_n^{\dagger_{csa}}\right)$$
$$(\,)^{\dagger_{csa}} = \mathbf{true} \qquad x^{\dagger_{csa}} = x \qquad n^{\dagger_{csa}} = n \qquad (t_1\ \mathtt{op}\ t_2)^{\dagger_{csa}} = t_1^{\dagger_{csa}}\ \mathtt{op}\ t_2^{\dagger_{csa}}$$
$$(\mathbf{if}\ p(t'_1, \ldots, t'_k)\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2)^{\dagger_{csa}} = (p(t'^{\dagger_{csa}}_1, \ldots, t'^{\dagger_{csa}}_k) \Rightarrow t_1^{\dagger_{csa}}) \wedge (\neg p(t'^{\dagger_{csa}}_1, \ldots, t'^{\dagger_{csa}}_k) \Rightarrow t_2^{\dagger_{csa}})$$
$$(\mathbf{event}\ a; t)^{\dagger_{csa}} = t^{\dagger_{csa}} \qquad (t_1\,t_2)^{\dagger_{csa}} = t_1^{\dagger_{csa}}\,t_2^{\dagger_{csa}} \qquad (t_1 \Box t_2)^{\dagger_{csa}} = t_1^{\dagger_{csa}} \wedge t_2^{\dagger_{csa}}.$$

Here, we assume that $\Omega(f_i) \geq \Omega(f_{i+1})$ for each $i \in \{1, \ldots, n-1\}$, and $\alpha_i = \nu$ if $\Omega(f_i)$ is even and μ otherwise.

The following theorem states the soundness and completeness of the reduction. See Appendix E.3 for a proof.

Theorem 4. *Let P be a program and Ω be a priority assignment for P. Then $\models_{csa} (P, \Omega)$ if and only if $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$.*

Example 9. Recall P_{app} and Ω_{app} above. We have $(P_{app}, \Omega_{app})^{\dagger_{csa}} = (\mathcal{E}, f_b\,5)$, where $\mathcal{E}$ is:

$$\mathtt{app}\,h\,x =_\mu h\,x; \quad f_b\,x =_\nu (x > 0 \Rightarrow \mathtt{app}\,f_a\,(x-1)) \wedge (x \leq 0 \Rightarrow \mathtt{app}\,f_b\,5);$$
$$f_a\,x =_\mu (x > 0 \Rightarrow \mathtt{app}\,f_a\,(x-1)) \wedge (x \leq 0 \Rightarrow \mathtt{app}\,f_b\,5).$$

Note that $\mathtt{L}_0 \models (P_{app}, \Omega_{app})^{\dagger_{csa}}$.

7.2 From temporal verification to call-sequence analysis

This subsection shows a reduction from the temporal verification problem **InfTraces**$(P) \cap L \stackrel{?}{=} \emptyset$ to a call-sequence analysis problem $\stackrel{?}{\models}_{csa} (P', \Omega)$.

For the sake of simplicity, we assume, without loss of generality,[11] that every program $P = (D, t)$ in this section is non-terminating and every infinite reduction sequence produces infinite events, so that $\mathbf{FullTraces}(P) = \mathbf{InfTraces}(P)$ holds. We also assume that the ω-regular language L for the temporal verification problem is specified by using a non-deterministic, parity word automaton [11]. We recall the definition of non-deterministic, parity word automata below.

Definition 9 (Parity automaton). *A* non-deterministic parity word automaton *(NPW)*[12] *is a quintuple* $\mathcal{A} = (Q, \Sigma, \delta, q_I, \Omega)$ *where (i)* Q *is a finite set of states; (ii)* Σ *is a finite alphabet; (iii)* δ*, called a transition function, is a* total *map from* $Q \times \Sigma$ *to* 2^Q*; (iv)* $q_I \in Q$ *is the initial state; and (v)* $\Omega : Q \to \{0, \ldots, p-1\}$ *is the priority function. A* run *of* $\mathcal{A}$ *on an* ω*-word* $a_0 a_1 \cdots \in \Sigma^\omega$ *is an infinite sequence of states* $\rho = \rho(0)\rho(1)\cdots \in Q^\omega$ *such that (i)* $\rho(0) = q_I$*, and (ii)* $\rho(i+1) \in \delta(\rho(i), a_i)$ *for each* $i \in \omega$*. An* ω*-word* $w \in \Sigma^\omega$ *is* accepted *by* $\mathcal{A}$ *if, there exists a run* ρ *of* $\mathcal{A}$ *on* w *such that* $\mathbf{max}\{\Omega(q) \mid q \in \mathbf{Inf}(\rho)\}$*is even, where* $\mathbf{Inf}(\rho)$ *is the set of states that occur infinitely often in* ρ*. We write* $\mathcal{L}(\mathcal{A})$ *for the set of* ω*-words accepted by* $\mathcal{A}$*.*

For technical convenience, we assume below that $\delta(q, a) \neq \emptyset$ for every $q \in Q$ and $a \in \Sigma$; this does not lose generality since if $\delta(q, a) = \emptyset$, we can introduce a new "dead" state q_{dead} (with priority 1) and change $\delta(q, a)$ to $\{q_{dead}\}$. Given a parity automaton A, we sometimes refer to each component of A by Q_A, Σ_A, δ_A, $q_{I,A}$ and Ω_A.

Example 10. Consider the automaton $\mathcal{A}_{ab} = (\{q_a, q_b\}, \{\mathtt{a}, \mathtt{b}\}, \delta, q_a, \Omega)$, where δ is as given in Figure 4, $\Omega(q_a) = 0$, and $\Omega(q_b) = 1$. Then, $\mathcal{L}(\mathcal{A}_{ab}) = \overline{(\mathtt{a}^*\mathtt{b})^\omega} = (\mathtt{a}^*\mathtt{b})^*\mathtt{a}^\omega$.

The goal of this subsection is, given a program P and a parity word automaton $\mathcal{A}$, to construct another program P' and a priority assignment Ω for P', such that $\mathbf{InfTraces}(P) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ if and only if $\models_{csa} (P', \Omega)$.

Note that a necessary and sufficient condition for $\mathbf{InfTraces}(P) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ is that no trace in $\mathbf{InfTraces}(P)$ has a run whose priority sequence satisfies the parity condition; in other words, for every sequence in $\mathbf{InfTraces}(P)$, and for every run for the sequence, the largest priority that occurs in the associated priority sequence is odd. As explained at the beginning of this section, we reduce

[11] As noted at the beginning of this section, every finite trace can be turned into an infinite trace by inserting (fresh) dummy events. Then, $\mathbf{InfTraces}(P) \cap L = \emptyset$ holds if and only if $\mathbf{InfTraces}(P') \cap L' = \emptyset$, where P' is the program obtained from P by inserting dummy events, and L' is the set of all event sequences obtained by inserting dummy events into a sequence in L.

[12] Note that non-deterministic Büchi automata may be viewed as instances of non-deterministic parity word automata, where there are only two priorities 1 and 2, and accepting and non-accepting states have priorities 2 and 1 respectively. We also note that the classes of deterministic parity, non-deteterministic parity, and non-deteterministic Büchi word automata accept the same class of ω-regular languages; here we opt for non-deteterministic parity word automata, because the translations from the others to NPW are trivial but the other directions may blow up the size of automata.

this condition to a call sequence analysis problem by appropriately duplicating functions in a given program. For example, recall the program P_0;

$$(\{f = \textbf{if } c \textbf{ then } (\textbf{event } \mathtt{a}; f) \textbf{ else } (\textbf{event } \mathtt{b}; f)\}, f)\,.$$

It is translated to P_0':

$$\begin{aligned}(\{&f_b = \textbf{if } c \textbf{ then } (\textbf{event } \mathtt{a}; f_a) \textbf{ else } (\textbf{event } \mathtt{b}; f_b),\\ &f_a = \textbf{if } c \textbf{ then } (\textbf{event } \mathtt{a}; f_a) \textbf{ else } (\textbf{event } \mathtt{b}; f_b)\}, f_b),\end{aligned}$$

where c is some (closed) boolean expression. Since the largest priorities encountered before calling f_a and f_b (since the last recursive call) respectively are 0 and 1, we assign those priorities plus 1 (to flip odd/even-ness) to f_a and f_b respectively. Then, the problem of $\textbf{InfTraces}(P_0) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ is reduced to $\models_{csa} (P_0', \{f_a \mapsto 1, f_b \mapsto 2\})$. Note here that the priorities of f_a and f_b represent *summaries* of the priorities (plus one) that occur in the run of the automaton until f_a and f_b are respectively called since the last recursive call; thus, the largest priority of states that occur infinitely often in the run for an infinite trace is equivalent to the largest priority that occurs infinitely often in the sequence of summaries $(\Omega(f_1) - 1)(\Omega(f_2) - 1)(\Omega(f_3) - 1) \cdots$ computed from a corresponding call sequence $f_1 f_2 f_3 \cdots$.

Due to the presence of higher-order functions, the general reduction is more complicated than the example above. First, we need to replicate not only function symbols, but also arguments. For example, consider the following variation P_1 of P_0 above:

$$(\{g\,k = \textbf{if } c \textbf{ then } (\textbf{event } \mathtt{a}; k) \textbf{ else } (\textbf{event } \mathtt{b}; k), \quad f = g\,f\}, \quad f)\,.$$

Here, we have just made the calls to f indirect, by preparing the function g. Obviously, the two calls to k in the body of g must be distinguished from each other, since different priorities are encountered before the calls. Thus, we duplicate the argument k, and obtain the following program P_1':

$$(\{g\,k_a\,k_b = \textbf{if } c \textbf{ then } (\textbf{event } \mathtt{a}; k_a) \textbf{ else } (\textbf{event } \mathtt{b}; k_b), \quad f_a = g\,f_a\,f_b, \quad f_b = g\,f_a\,f_b\}, \quad f_a)\,.$$

Then, for the priority assignment $\Omega = \{f_a \mapsto 1, f_b \mapsto 2, g \mapsto 1\}$, $\textbf{InfTraces}(P_1) \cap \mathcal{L}(\mathcal{A}_{ab}) = \emptyset$ if and only if $\models_{csa} (P_1', \Omega)$. Secondly, we need to take into account not only the priorities of states visited by $\mathcal{A}$, but also the states themselves. For example, if we have a function definition $f\,h = h(\textbf{event } \mathtt{a}; f\,h)$, the largest priority encountered before f is recursively called in the body of f depends on the priorities encountered inside h, *and also* the state of $\mathcal{A}$ when h uses the argument **event** a; f (because the state after the a event depends on the previous state in general). We, therefore, use *intersection types* (a la Kobayashi and Ong's intersection types for HORS model checking [22]) to represent summary information on how each function traverses states of the automaton, and replicate each function and its arguments for each type. We thus formalize the translation as an intersection-type-based program transformation; related transformation techniques are found in [9, 12, 13, 21, 39].

Definition 10. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, \Omega)$ be a non-deterministic parity word automaton. Let q and m range over Q and the set codom(Ω) of priorities respectively. The set* $\mathbf{Types}_{\mathcal{A}}$ *of* intersection types*, ranged over by θ, is defined by:*

$$\theta ::= q \mid \rho \to \theta \qquad\qquad \rho ::= \mathtt{int} \mid \textstyle\bigwedge_{1\le i\le k}(\theta_i, m_i)$$

We assume a certain total order $<$ on $\mathbf{Types}_{\mathcal{A}} \times \mathbb{N}$, and require that in $\bigwedge_{1\le i\le k}(\theta_i, m_i)$, $(\theta_i, m_i) < (\theta_j, m_j)$ holds for each $i < j$.

We often write $(\theta_1, m_1) \wedge \cdots (\theta_k, m_k)$ for $\bigwedge_{1\le i\le k}(\theta_i, m_i)$, and $\top$ when $k = 0$. Intuitively, the type q describes expressions of simple type $\star$, which may be evaluated when the automaton $\mathcal{A}$ is in the state q (here, we have in mind an execution of the *product* of a program and the automaton, where the latter takes events produced by the program and changes its states). The type $(\bigwedge_{1\le i\le k}(\theta_i, m_i)) \to \theta$ describes functions that take an argument, use it according to types $\theta_1, \ldots, \theta_k$, and return a value of type θ. Furthermore, the part m_i describes that the argument may be used as a value of type θ_i only when the largest priority visited since the function is called is m_i. For example, given the automaton in Example 10, the function $\lambda x.(\textbf{event}\ \mathtt{a}; x)$ may have types $(q_a, 0) \to q_a$ and $(q_a, 0) \to q_b$, because the body may be executed from state q_a or q_b (thus, the return type may be any of them), but x is used only when the automaton is in state q_a and the largest priority visited is 1. In contrast, $\lambda x.(\textbf{event}\ \mathtt{b}; x)$ have types $(q_b, 1) \to q_a$ and $(q_b, 1) \to q_b$.

Using the intersection types above, we shall define a type-based transformation relation of the form $\Gamma \vdash_{\mathcal{A}} t : \theta \Rightarrow t'$, where t and t' are the source and target terms of the transformation, and Γ, called an *intersection type environment*, is a finite set of type bindings of the form $x : \mathtt{int}$ or $x : (\theta, m, m')$. We allow multiple type bindings for a variable x except for $x : \mathtt{int}$ (i.e. if $x : \mathtt{int} \in \Gamma$, then this must be the unique type binding for x in Γ). The binding $x : (\theta, m, m')$ means that x should be used as a value of type θ when the largest priority visited is m; m' is auxiliary information used to record the largest priority encountered so far.

The transformation relation $\Gamma \vdash_{\mathcal{A}} t : \theta \Rightarrow t'$ is inductively defined by the rules in Figure 5. (For technical convenience, we have extended terms with λ-abstractions; they may occur only at top-level function definitions.) The operation $\Gamma \uparrow m$ used in the figure is defined by:

$$\Gamma \uparrow m = \{x : \mathtt{int} \mid x : \mathtt{int} \in \Gamma\} \cup \{x : (\theta, m_1, \mathbf{max}(m_2, m)) \mid x : (\theta, m_1, m_2) \in \Gamma\}$$

The operation is applied when the priority m is encountered, in which case the largest priority encountered is updated accordingly. The key rules are IT-Var, IT-Event, IT-App, and IT-Abs. In IT-Var, the variable x is replicated for each type; in the target of the translation, $x_{\theta,m}$ and $x_{\theta',m'}$ are treated as different variables if $(\theta, m) \neq (\theta', m')$. The rule IT-Event reflects the state change caused by the event a to the type and the type environment. Since the state change may be non-deterministic, we transform t for each of the next states $q_1, \ldots, q_n$, and combine the resulting terms with non-deterministic choice. The rule IT-App and IT-Abs replicates function arguments for each type. In addition, in IT-App, the operation

$$\frac{}{\Gamma \vdash_{\mathcal{A}} () : q \Rightarrow ()} \quad \text{(IT-Unit)}$$

$$\frac{}{\Gamma, x : \texttt{int} \vdash_{\mathcal{A}} x : \texttt{int} \Rightarrow x_{\texttt{int}}} \quad \text{(IT-VarInt)}$$

$$\frac{}{\Gamma, x : (\theta, m, m) \vdash_{\mathcal{A}} x : \theta \Rightarrow x_{\theta,m}} \quad \text{(IT-Var)}$$

$$\frac{}{\Gamma \vdash_{\mathcal{A}} n : \texttt{int} \Rightarrow n} \quad \text{(IT-Int)}$$

$$\frac{\Gamma \vdash_{\mathcal{A}} t_1 : \texttt{int} \Rightarrow t_1' \qquad \Gamma \vdash_{\mathcal{A}} t_2 : \texttt{int} \Rightarrow t_2'}{\Gamma \vdash_{\mathcal{A}} t_1 \;\texttt{op}\; t_2 : \texttt{int} \Rightarrow t_1' \;\texttt{op}\; t_2'} \quad \text{(IT-Op)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{A}} t_i : \texttt{int} \Rightarrow t_i' \quad (\text{for each } i \in \{1, \ldots, n\}) \\ \Gamma \vdash_{\mathcal{A}} t_{n+1} : q \Rightarrow t_{n+1}' \qquad \Gamma \vdash_{\mathcal{A}} t_{n+2} : q \Rightarrow t_{n+2}'\end{array}}{\begin{array}{c}\Gamma \vdash_{\mathcal{A}} \textbf{if } p(t_1, \ldots, t_n) \textbf{ then } t_{n+1} \textbf{ else } t_{n+2} : q \\ \Rightarrow \textbf{if } p(t_1', \ldots, t_n') \textbf{ then } t_{n+1}' \textbf{ else } t_{n+2}'\end{array}} \quad \text{(IT-If)}$$

$$\frac{\begin{array}{c}\delta_A(q, a) = \{q_1, \ldots, q_n\} \\ \Gamma \uparrow \Omega_A(q_i) \vdash_{\mathcal{A}} t : q_i \Rightarrow t_i' \quad (\text{for each } i \in \{1, \ldots, n\})\end{array}}{\Gamma \vdash_{\mathcal{A}} (\textbf{event } a; t) : q \Rightarrow (\textbf{event } a; t_1' \Box \cdots \Box t_n')} \quad \text{(IT-Event)}$$

$$\frac{\Gamma \vdash_{\mathcal{A}} t_1 : q \Rightarrow t_1' \qquad \Gamma \vdash_{\mathcal{A}} t_2 : q \Rightarrow t_2'}{\Gamma \vdash_{\mathcal{A}} t_1 \Box t_2 : q \Rightarrow t_1' \Box t_2'} \quad \text{(IT-NonDet)}$$

$$\frac{\Gamma \vdash_{\mathcal{A}} t_1 : \texttt{int} \to \theta \Rightarrow t_1' \qquad \Gamma \vdash_{\mathcal{A}} t_2 : \texttt{int} \Rightarrow t_2'}{\Gamma \vdash_{\mathcal{A}} t_1\, t_2 : \theta \Rightarrow t_1'\, t_2'} \quad \text{(IT-AppInt)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{A}} t_1 : \bigwedge_{1 \le i \le k} (\theta_i, m_i) \to \theta \Rightarrow t_1' \\ \Gamma \uparrow m_i \vdash_{\mathcal{A}} t_2 : \theta_i \Rightarrow t_{2,i}' \ (\text{for each } i \in \{1, \ldots, k\})\end{array}}{\Gamma \vdash_{\mathcal{A}} t_1\, t_2 : \theta \Rightarrow t_1'\, t_{2,1}' \ \ldots\ t_{2,k}'} \quad \text{(IT-App)}$$

$$\frac{\Gamma, x : \texttt{int} \vdash_{\mathcal{A}} t : \theta \Rightarrow t' \qquad x \notin \mathit{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{A}} \lambda x.t : \texttt{int} \to \theta \Rightarrow \lambda x_{\texttt{int}}.t'} \quad \text{(IT-AbsInt)}$$

$$\frac{\begin{array}{c}\Gamma, x : (\theta_1, m_1, 0), \ldots, x : (\theta_k, m_k, 0) \vdash_{\mathcal{A}} t : \theta' \Rightarrow t' \\ x \notin \mathit{dom}(\Gamma)\end{array}}{\begin{array}{c}\Gamma \vdash_{\mathcal{A}} \lambda x.t : \bigwedge_{1 \le i \le k} (\theta_i, m_i) \to \theta' \\ \Rightarrow \lambda x_{\theta_1, m_1} \ldots x_{\theta_k, m_k}.t'\end{array}} \quad \text{(IT-Abs)}$$

Fig. 5. Type-based Transformation Rules for Terms

$\Gamma \uparrow m_i$ reflects the fact that t_2 is used as a value of type θ_i after the priority m_i is encountered. The other rules just transform terms in a compositional manner. If target terms are ignored, the entire rules are close to those of Kobayashi and Ong's type system for HORS model checking [22].

We now define the transformation for programs. A *top-level type environment* Ξ is a finite set of type bindings of the form $x : (\theta, m)$. Like intersection type environments, Ξ may have more than one binding for each variable. We write $\Xi \vdash_{\mathcal{A}} t : \theta$ to mean $\{x : (\theta, m, 0) \mid x : (\theta, m) \in \Xi\} \vdash_{\mathcal{A}} t : \theta$. For a set D of function definitions, we write $\Xi \vdash_{\mathcal{A}} D \Rightarrow D'$ if $\mathit{dom}(D') = \{ f_{\theta,m} \mid f : (\theta, m) \in \Xi \}$ and $\Xi \vdash_{\mathcal{A}} D(f) : \theta \Rightarrow D'(f_{\theta,m})$ for every $f : (\theta, m) \in \Xi$. For a program $P = (D, t)$, we write $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega')$ if $P' = (D', t')$, $\Xi \vdash_{\mathcal{A}} D \Rightarrow D'$ and $\Xi \vdash_{\mathcal{A}} t : q_I \Rightarrow t'$, with $\Omega'(f_{\theta,m}) = m + 1$ for each $f_{\theta,m} \in \mathit{dom}(D')$. We just write $\vdash_{\mathcal{A}} P \Rightarrow (P', \Omega')$ if $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega')$ holds for some Ξ.

Example 11. Consider the automaton $\mathcal{A}_{ab}$ in Example 10, and the program $P_2 = (D_2, f\,5)$ where D_2 consists of the following function definitions:

$$g\ k = (\textbf{event}\ \texttt{a}; k) \Box (\textbf{event}\ \texttt{b}; k), \quad f\ x = \textbf{if}\ x > 0\ \textbf{then}\ g\,(f(x-1))\ \textbf{else}\ (\textbf{event}\ \texttt{b}; f\,5).$$

Let Ξ be:

$$g : ((q_a, 0) \wedge (q_b, 1) \to q_a, 0), \quad g : ((q_a, 0) \wedge (q_b, 1) \to q_b, 0),$$
$$f : (\texttt{int} \to q_a, 0), \quad f : (\texttt{int} \to q_a, 0), \quad f : (\texttt{int} \to q_b, 1)$$

Then, $\Xi \vdash_{\mathcal{A}} P_1 \Rightarrow ((D'_2, f_{\texttt{int}\to q_a,0}\, 5), \Omega)$ where:

$$\begin{aligned}
D'_2 &= \{g_{(q_a,0)\wedge(q_b,1)\to q_a,0}\; k_{q_a,0}\; k_{q_b,1} = t_g, \quad g_{(q_a,0)\wedge(q_b,1)\to q_b,0}\; k_{q_a,0}\; k_{q_b,1} = t_g,\\
&\qquad f_{\texttt{int}\to q_a,0}\; x_{\texttt{int}} = t_{f,q_a}, \quad f_{\texttt{int}\to q_b,1}\; x_{\texttt{int}} = t_{f,q_b}\}\\
t_g &= (\textbf{event}\ \mathsf{a}; k_{q_a,0})\Box(\textbf{event}\ \mathsf{b}; k_{q_b,1}),\\
t_{f,q} &= \textbf{if}\ x_{\texttt{int}} > 0\ \textbf{then}\ g_{(q_a,0)\wedge(q_b,1)\to q,0}\,(f_{\texttt{int}\to q_a,0}(x_{\texttt{int}}-1))\,(f_{\texttt{int}\to q_b,1}(x_{\texttt{int}}-1))\\
&\qquad \textbf{else}\ (\textbf{event}\ \mathsf{b}; f_{\texttt{int}\to q_b,1}\, 5), \qquad\qquad (q \in \{q_a, q_b\})\\
\Omega &= \{g_{(q_a,0)\wedge(q_b,1)\to q_a,0} \mapsto 1, g_{(q_a,0)\wedge(q_b,1)\to q_b,0} \mapsto 1, f_{\texttt{int}\to q_a,0} \mapsto 1, f_{\texttt{int}\to q_b,1} \mapsto 2\}
\end{aligned}$$

Appendix F shows how t_g and t_f are derived. Notice that f, g, and the arguments of g have been duplicated. Furthermore, whenever $f_{\theta,m}$ is called, the largest priority that has been encountered since the last recursive call is m. For example, in the then-clause of $f_{\texttt{int}\to q_a,0}$, $f_{\texttt{int}\to q_b,1}(x-1)$ may be called through $g_{(q_a,0)\wedge(q_b,1)\to q_a,0}$. Since $g_{(q_a,0)\wedge(q_b,1)\to q_a,0}$ uses the second argument only after an event b, the largest priority encountered is 1. This property is important for the correctness of our reduction.

The following theorem claims the soundness and completeness of our reduction. See Appendix E for a proof.

Theorem 5. *Let P be a program and $\mathcal{A}$ be a parity automaton. Suppose that $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega)$. Then* $\textbf{InfTraces}(P) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ *if and only if* $\models_{csa} (P', \Omega)$.

Furthermore, one can effectively find an appropriate transformation. See Appendix E.5 for a proof sketch.

Theorem 6. *For every P and $\mathcal{A}$, one can effectively construct Ξ, P' and Ω such that $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega)$.*

For an order-k program P, the size of P' is polynomial in the size of P if the other parameters, i.e., the size of $\mathcal{A}$ and the largest arity of functions in P are fixed, but the worst-case size of P' can be k-fold exponential in those parameters. As in HORS model checking [7, 19, 34], however, we expect that the worst-case is not so often encountered in realistic verification problems.

Combined with the reduction from call-sequence analysis to HFL model checking, we have thus obtained an effective reduction from the temporal verification problem $\textbf{InfTraces}(P) \overset{?}{\subseteq} \mathcal{L}(\mathcal{A})$ to HFL model checking.

8 Related Work

As mentioned in Section 1, our reduction from program verification problems to HFL model checking has been partially inspired by the translation of Kobayashi et al. [20] from HORS model checking to HFL model checking. As in their translation (and unlike in previous applications of HFL model checking [28, 43]), our translation switches the roles of properties and models (or programs) to be verified. Although a combination of their translation with Kobayashi's reduction from program verification to HORS model checking [18, 19] yields an (indirect) translation from *finite-data* programs to pure HFL model checking problems, the combination does not work for infinite-data programs. In contrast, our translation is sound and complete even for infinite-data programs. Among the translations in Sections 5–7, the translation in Section 7.2 shares some similarity to their translation, in that functions and their arguments are replicated for each priority. The actual translations are however quite different; ours is type-directed and optimized for a given automaton, whereas their translation is not. This difference comes from the difference of the goals: the goal of [20] was to clarify the relationship between HORS and HFL, hence their translation was designed to be independent of an automaton. The proof of the correctness of our translation in Section 7 is much more involved (cf. Appendix D and E), due to the need for dealing with integers. Whilst the proof of [20] could reuse the type-based characterization of HORS model checking [22], we had to generalize arguments in both [22] and [20] to work on infinite-data programs.

Lange et al. [28] have shown that various process equivalence checking problems (such as bisimulation and trace equivalence) can be reduced to (pure) HFL model checking problems. The idea of their reduction is quite different from ours. They reduce processes to LTSs, whereas we reduce programs to HFL formulas.

Major approaches to automated or semi-automated higher-order program verification have been HORS model checking [18, 19, 23, 27, 31, 33, 44], (refinement) type systems [15, 24, 35–37, 40, 42, 45], Horn clause solving [2, 8], and their combinations. As already discussed in Section 1, compared with the HORS model checking approach, our new approach provides more uniform, streamlined methods. Whilst the HORS model checking approach is for fully automated verification, our approach enables various degrees of automation: after verification problems are automatically translated to $\mathrm{HFL}_{\mathbf{Z}}$ formulas, one can prove them (i) interactively using a proof assistant like Coq (see Appendix G), (ii) semi-automatically, by letting users provide hints for induction/co-induction and discharging the rest of proof obligations by (some extension of) an SMT solver, or (iii) fully automatically by recasting the techniques used in the HORS-based approach; for example, to deal with the ν-only fragment of $\mathrm{HFL}_{\mathbf{Z}}$, we can reuse the technique of predicate abstraction [23]. For a more technical comparison between the HORS-based approach and our HFL-based approach, see Appendix H.

As for type-based approaches [15, 24, 35–37, 40, 42, 45], most of the refinement type systems are (i) restricted to safety properties, and/or (ii) incomplete. A notable exception is the recent work of Unno et al. [41], which provides a relatively complete type system for the classes of properties discussed in Section 5. Our approach deals with a wider class of properties (cf. Sections 6 and 7). Their "relative completeness" property relies on Godel coding of functions, which cannot be exploited in practice.

The reductions from program verification to Horn clause solving have recently been advocated [2–4] or used [35, 40] (via refinement type inference problems) by a number of researchers. Since Horn clauses can be expressed in a fragment of HFL without modal operators, fixpoint alternations (between ν and μ), and higher-order predicates, our reductions to HFL model checking may be viewed as extensions of those approaches. Higher-order predicates and fixpoints over them allowed us to provide sound and complete characterizations of properties of higher-order programs for a wider class of properties. Bjørner et al. [4] proposes an alternative approach to obtaining a complete characterization of safety properties, which defunctionalizes higher-order programs by using algebraic data types and then reduces the problems to (first-order) Horn clauses. A disadvantage of that approach is that control flow information of higher-order programs is also encoded into algebraic data types; hence even for finite-data higher-order programs, the Horn clauses obtained by the reduction belong to an undecidable fragment. In contrast, our reductions yield pure HFL model checking problems for finite-data programs. Burn et al. [8] have recently advocated the use of *higher-order* (constrained) Horn clauses for verification of safety properties (i.e., which correspond to the negation of may-reachability properties discussed in Section 5.1 of the present paper) of higher-order programs. They interpret recursion using the least fixpoint semantics, so their higher-order Horn clauses roughly corresponds to a fragment of the $\mathrm{HFL}_{\mathbf{Z}}$ without modal operators and fixpoint alternations. They have not shown a general, concrete reduction from safety property verification to higher-order Horn clause solving (see the last paragraph of the conclusion of [8]).

The characterization of the reachability problems in Section 5 in terms of formulas without modal operators is a reminiscent of predicate transformers [10, 14] used for computing the weakest preconditions of imperative programs. In particular, [5] and [14] respectively used least fixpoints to express weakest preconditions for while-loops and recursions.

9 Conclusion

We have shown that various verification problems for higher-order functional programs can be naturally reduced to (extended) HFL model checking problems. In all the reductions, a program is mapped to an HFL formula expressing the property that the behavior of the program is correct. For developing verification tools for higher-order functional programs, our reductions allow us to focus on the development of (automated or semi-automated) $\mathrm{HFL}_{\mathbf{Z}}$ model checking tools (or, even more simply, theorem provers for $\mathrm{HFL}_{\mathbf{Z}}$ without modal operators, as the reductions of Section 5 and 7 yield HFL formulas without modal operators). To this end, we have developed a prototype model checker for pure HFL (without integers), which will be reported in a separate paper. Work is under way to develop $\mathrm{HFL}_{\mathbf{Z}}$ model checkers by recasting the techniques [23, 26, 27, 44] developed for the HORS-based approach, which, together with the reductions presented in this paper, would yield fully automated verification tools. We have also started building a Coq library for interactively proving $\mathrm{HFL}_{\mathbf{Z}}$ formulas, as briefly discussed in Appendix G. As a final remark, although one may fear that our reductions may map program verification problems to "harder" problems due to the expressive power of $\text{HFL}_{\mathbf{Z}}$, it is actually not the case at least for the classes of problems in Section 5 and 6, which use the only alternation-free fragment of $\text{HFL}_{\mathbf{Z}}$. The model checking problems for μ-only or ν-only $\text{HFL}_{\mathbf{Z}}$ are semi-decidable and co-semi-decidable respectively, like the source verification problems of may/must-reachability and their negations of closed programs.

References

1. Axelsson, R., Lange, M., Somla, R.: The complexity of model checking higher-order fixpoint logic. Logical Methods in Computer Science 3(2) (2007)
2. Bjørner, N., Gurfinkel, A., McMillan, K.L., Rybalchenko, A.: Horn clause solvers for program verification. LNCS, vol. 9300, pp. 24–51. Springer (2015)
3. Bjørner, N., McMillan, K.L., Rybalchenko, A.: Program verification as satisfiability modulo theories. In: SMT 2012. pp. 3–11. EasyChair (2012)
4. Bjørner, N., McMillan, K.L., Rybalchenko, A.: Higher-order program verification as satisfiability modulo theories with algebraic data-types. CoRR abs/1306.5264 (2013)
5. 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)
6. Blume, M., Acar, U.A., Chae, W.: Exception handlers as extensible cases. In: Proceedings of APLAS 2008. LNCS, vol. 5356, pp. 273–289. Springer (2008)
7. Broadbent, C.H., Kobayashi, N.: Saturation-based model checking of higher-order recursion schemes. In: Proceedings of CSL 2013. LIPIcs, vol. 23, pp. 129–148 (2013)
8. Burn, T.C., Ong, C.L., Ramsay, S.J.: Higher-order constrained horn clauses and refinement types. CoRR abs/1705.06216 (2017)
9. Carayol, A., Serre, O.: Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In: LICS 2012. pp. 165–174. IEEE (2012)
10. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM 18(8), 453–457 (1975)
11. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, LNCS, vol. 2500. Springer (2002)
12. Grellois, C., Melliès, P.: Relational semantics of linear logic and higher-order model checking. In: Proceedings of CSL 2015. LIPIcs, vol. 41, pp. 260–276 (2015)
13. Haddad, A.: Model checking and functional program transformations. In: Proceedings of FSTTCS 2013. LIPIcs, vol. 24, pp. 115–126 (2013)
14. Hesselink, W.H.: Predicate-transformer semantics of general recursion. Acta Inf. 26(4), 309–332 (1989)
15. Hofmann, M., Chen, W.: Abstract interpretation from Büchi automata. In: Proceedings of CSL-LICS ’14. pp. 51:1–51:10. ACM (2014)
16. Igarashi, A., Kobayashi, N.: Resource usage analysis. ACM Trans. Prog. Lang. Syst. 27(2), 264–313 (2005)
17. Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: FoSSaCS 2002. LNCS, vol. 2303, pp. 205–222. Springer (2002)
18. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: Proceedings of POPL. pp. 416–428. ACM Press (2009)
19. Kobayashi, N.: Model checking higher-order programs. Journal of the ACM 60(3) (2013)
20. Kobayashi, N., Lozes, É., Bruse, F.: On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In: Proceedings of POPL 2017. pp. 246–259 (2017)
21. Kobayashi, N., Matsuda, K., Shinohara, A., Yaguchi, K.: Functional programs as compressed data. Higher-Order and Symbolic Computation (2013)
22. Kobayashi, N., Ong, C.H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: Proceedings of LICS 2009. pp. 179–188 (2009)

23. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Proc. of PLDI. pp. 222–233. ACM Press (2011)
24. Koskinen, E., Terauchi, T.: Local temporal reasoning. In: Proceedings of CSL-LICS '14. pp. 59:1–59:10. ACM (2014)
25. Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. 27, 333–354 (1983)
26. Kuwahara, T., Sato, R., Unno, H., Kobayashi, N.: Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In: Proceedings of CAV 2015. LNCS, vol. 9207, pp. 287–303. Springer (2015)
27. Kuwahara, T., Terauchi, T., Unno, H., Kobayashi, N.: Automatic termination verification for higher-order functional programs. In: Proceedings of ESOP 2014. Lecture Notes in Computer Science, vol. 8410, pp. 392–411. Springer (2014)
28. Lange, M., Lozes, É., Guzmán, M.V.: Model-checking process equivalences. Theor. Comput. Sci. 560, 326–347 (2014)
29. Ledesma-Garza, R., Rybalchenko, A.: Binary reachability analysis of higher order functional programs. In: SAS 2012. LNCS, vol. 7460, pp. 388–404. Springer (2012)
30. Lozes, É.: A type-directed negation elimination. In: Proceedings FICS 2015. EPTCS, vol. 191, pp. 132–142 (2015)
31. Murase, A., Terauchi, T., Kobayashi, N., Sato, R., Unno, H.: Temporal verification of higher-order functional programs. In: Proceedings of POPL 2016. pp. 57–68 (2016)
32. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS 2006. pp. 81–90. IEEE Computer Society Press (2006)
33. Ong, C.H.L., Ramsay, S.: Verifying higher-order programs with pattern-matching algebraic data types. In: Proceedings of POPL. pp. 587–598. ACM Press (2011)
34. Ramsay, S., Neatherway, R., Ong, C.H.L.: An abstraction refinement approach to higher-order model checking. In: Proceedings of POPL 2014. pp. 61–72. ACM (2014)
35. Rondon, P.M., Kawaguchi, M., Jhala, R.: Liquid types. In: PLDI 2008. pp. 159–169 (2008)
36. Skalka, C., Smith, S.F., Horn, D.V.: Types and trace effects of higher order programs. J. Funct. Program. 18(2), 179–249 (2008)
37. Terauchi, T.: Dependent types from counterexamples. In: Proceedings of POPL. pp. 119–130. ACM (2010)
38. Tobita, Y., Tsukada, T., Kobayashi, N.: Exact flow analysis by higher-order model checking. In: Proceedings of FLOPS 2012. LNCS, vol. 7294, pp. 275–289. Springer (2012)
39. Tsukada, T., Ong, C.L.: Compositional higher-order model checking via ω-regular games over böhm trees. In: Proceedings of CSL-LICS '14. pp. 78:1–78:10. ACM (2014)
40. Unno, H., Kobayashi, N.: Dependent type inference with interpolants. In: PPDP 2009. pp. 277–288. ACM (2009)
41. Unno, H., Satake, Y., Terauchi, T.: Relatively complete refinement type system for verification of higher-order non-deterministic programs. In: Proceeedings of POPL 2018 (2018), to appear
42. Unno, H., Terauchi, T., Kobayashi, N.: Automating relatively complete verification of higher-order functional programs. In: POPL 2013. pp. 75–86. ACM (2013)
43. 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)
44. Watanabe, K., Sato, R., Tsukada, T., Kobayashi, N.: Automatically disproving fair termination of higher-order functional programs. In: Proceedings of ICFP 2016. pp. 243–255. ACM (2016)
45. Zhu, H., Nori, A.V., Jagannathan, S.: Learning refinement types. In: Proceedings of ICFP 2015. pp. 400–411. ACM (2015)

Appendix

A Typing Rules for Programs

The type judgments for expressions and programs are of the form $\mathcal{K} \vdash t : \eta$ and $\mathcal{K} \vdash P$, where $\mathcal{K}$ is a finite map from variables to types. The typing rules are shown in Figure 6. We write $\vdash P$ if $\mathcal{K} \vdash P$ for some $\mathcal{K}$.

$$\frac{}{\mathcal{K} \vdash () : \star} \qquad \frac{\mathtt{arity}(p) = k \quad \mathcal{K} \vdash t'_i : \mathtt{int} \text{ for each } i \in \{1, \ldots, k\} \quad \mathcal{K} \vdash t_j : \star \text{ for each } j \in \{1, 2\}}{\mathcal{K} \vdash \mathbf{if}\ p(t'_1, \ldots, t'_k)\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 : \star}$$

$$\frac{}{\mathcal{K}, x : \eta \vdash x : \eta} \qquad \frac{\mathcal{K} \vdash t_1 : \eta \to \kappa \quad \mathcal{K} \vdash t_2 : \eta}{\mathcal{K} \vdash t_1 t_2 : \kappa}$$

$$\frac{}{\mathcal{K} \vdash n : \mathtt{int}} \qquad \frac{\mathcal{K} \vdash t_1 : \star \quad \mathcal{K} \vdash t_2 : \star}{\mathcal{K} \vdash t_1 \Box t_2 : \star}$$

$$\frac{\mathcal{K} \vdash t_1 : \mathtt{int} \quad \mathcal{K} \vdash t_2 : \mathtt{int}}{\mathcal{K} \vdash t_1\ \mathtt{op}\ t_2 : \mathtt{int}}$$

$$\frac{\mathcal{K} \vdash t : \star}{\mathcal{K} \vdash \mathbf{event}\ a; t : \star} \qquad \frac{\mathcal{K} = f_1 : \kappa_1, \ldots, f_n : \kappa_n \quad \mathcal{K} \vdash t : \star \quad \mathcal{K}, \widetilde{x_i} : \widetilde{\eta_i} \vdash t_i : \star \quad \kappa_i = \widetilde{\eta_i} \to \star \text{ for each } i \in \{1, \ldots, n\}}{\mathcal{K} \vdash (\{f_1\ \widetilde{x_1} = t_1, \ldots, f_n\ \widetilde{x_n} = t_n\}, t)}$$

Fig. 6. Typing Rules for Expressions and Programs

B Proofs for Section 5

B.1 Proofs for Section 5.1

To prove the theorem, we define the reduction relation $t \longrightarrow_D t'$ as given in Figure 7. It differs from the labeled transition semantics in that $\Box$ and **event** $a; \cdots$ are not eliminated; this semantics is more convenient for establishing the relationship between a program and a corresponding $\mathrm{HFL}_{\mathbf{Z}}$ formula. It should be clear that $a \in \mathbf{Traces}(D, t)$ if and only if $t \longrightarrow_D^* E[\mathbf{event}\ a; t']$ for some t'.

We shall first prove the theorem for recursion-free programs. Here, a program $P = (D, t)$ is *recursion-free* if the transitive closure of the relation $\{(f_i, f_j) \in dom(D) \times dom(D) \mid f_j \text{ occurs in } D(f_i)\}$ is irreflexive. To this end, we prepare a few lemmas.

The following lemma says that the semantics of $\mathrm{HFL}_{\mathbf{Z}}$ formulas is preserved by reductions of the corresponding programs.

Lemma 1. *Let (D, t) be a program and* L *be an LTS. If $t \longrightarrow_D t'$, then* $[\![(D, t)^{\dagger_{may}}]\!]_{\mathtt{L}} = [\![(D, t')^{\dagger_{may}}]\!]_{\mathtt{L}}$.

$$E(\text{evaluation contexts}) ::= [\,] \mid E\Box t \mid t\Box E \mid \textbf{event}\ a; E$$

$$\frac{f\,\widetilde{x} = u \in D \qquad |\widetilde{x}| = |\widetilde{t}|}{E[f\,\widetilde{t}] \longrightarrow_D E[[\widetilde{t}/\widetilde{x}]u]}\ (\text{R-Fun}) \qquad \frac{([\![t'_1]\!], \dots, [\![t'_k]\!]) \notin [\![p]\!]}{E[\textbf{if}\ p(t'_1, \dots, t'_k)\ \textbf{then}\ t_1\ \textbf{else}\ t_2] \longrightarrow_D E[t_2]}\ (\text{R-IfF})$$

$$\frac{([\![t'_1]\!], \dots, [\![t'_k]\!]) \in [\![p]\!]}{E[\textbf{if}\ p(t'_1, \dots, t'_k)\ \textbf{then}\ t_1\ \textbf{else}\ t_2] \longrightarrow_D E[t_1]}\ (\text{R-IfT})$$

Fig. 7. Reduction Semantics

Proof. Let $D = \{f_1\,\widetilde{x}_1 = t_1, \dots, f_n\,\widetilde{x}_n = t_n\}$, and $(F_1, \dots, F_n)$ be the least fixpoint of

$$\lambda(X_1, \dots, X_n).([\![\lambda\widetilde{x}_1.t_1{}^{\dagger_{may}}]\!](\{f_1 \mapsto X_1, \dots, f_n \mapsto X_n\}), \dots, [\![\lambda\widetilde{x}_n.t_n{}^{\dagger_{may}}]\!](\{f_1 \mapsto X_1, \dots, f_n \mapsto X_n\})).$$

By the Bekić property, $[\![(D,t)^{\dagger_{may}}]\!] = [\![t^{\dagger_{may}}]\!](\{f_1 \mapsto F_1, \dots, f_n \mapsto F_n\})$. Thus, it suffices to show that $t \longrightarrow_D t'$ implies $[\![t^{\dagger_{may}}]\!](\rho) = [\![t'^{\dagger_{may}}]\!](\rho)$ for $\rho = \{f_1 \mapsto F_1, \dots, f_n \mapsto F_n\}$. We show it by case analysis on the rule used for deriving $t \longrightarrow_D t'$.

- Case R-Fun: In this case, $t = E[f_i\,\widetilde{s}]$ and $t' = E[[\widetilde{s}/\widetilde{x}_i]t_i]$. Since $(F_1, \dots, F_n)$ is a fixpoint, we have:
$$\begin{aligned}&[\![f_i\,\widetilde{s}]\!](\rho) = F_i([\![\widetilde{s}]\!](\rho))\\ &= [\![\lambda\widetilde{x}_i.t_i]\!](\rho)([\![\widetilde{s}]\!](\rho))\\ &= [\![[\widetilde{s}/\widetilde{x}_i]t_i]\!](\rho)\end{aligned}$$
Thus, we have $[\![t^{\dagger_{may}}]\!](\rho) = [\![t'^{\dagger_{may}}]\!](\rho)$ as required.
- Case R-IfT: In this case, $t = E[\textbf{if}\ p(s'_1, \dots, s'_k)\ \textbf{then}\ s_1\ \textbf{else}\ s_2]$ and $t' = E[s_1]$ with $([\![s'_1]\!], \dots, [\![s'_k]\!]) \in [\![p]\!]$. Thus, $t^{\dagger_{may}} = E^{\dagger_{may}}[(p(s'_1, \dots, s'_k) \wedge s_1{}^{\dagger_{may}}) \vee (\neg p(s'_1, \dots, s'_k) \wedge s_2{}^{\dagger_{may}})]$. Since $([\![s'_1]\!], \dots, [\![s'_k]\!]) \in [\![p]\!]$, $([\![s'_1]\!], \dots, [\![s'_k]\!]) \notin [\![\neg p]\!]$. Thus, $[\![t^{\dagger_{may}}]\!](\rho) = [\![E^{\dagger_{may}}[(\textbf{true} \wedge s_1{}^{\dagger_{may}}) \vee (\textbf{false} \wedge s_2{}^{\dagger_{may}})]]\!](\rho) = [\![E^{\dagger_{may}}[s_1{}^{\dagger_{may}}]]\!](\rho)$. We have thus $[\![t^{\dagger_{may}}]\!](\rho) = [\![t'^{\dagger_{may}}]\!](\rho)$ as required.
- Case R-IfF: Similar to the above case.

□

The following lemma says that Theorem 1 holds for programs in normal form.

Lemma 2. *Let (D,t) be a program and $t \not\longrightarrow_D$. Then, $\mathsf{L}_0 \models (D,t)^{\dagger_{may}}$ if and only if $t = E[\textbf{event}\ a; t']$ for some evaluation context E and t'.*

Proof. The proof proceeds by induction on the structure of t. By the condition $t \not\longrightarrow_D$ and the (implicit) assumption that $\vdash (D,t)$, t is generated by the following grammar:

$$t ::= (\,) \mid \textbf{event}\ a; t' \mid t_1 \Box t_2.$$

– Case $t = ()$: The result follows immediately, as t is not of the form $E[\textbf{event}\ a; t']$, and $t^{\dagger_{may}} = \textbf{false}$.
– Case $t = \textbf{event}\ a; t'$: The result follows immediately, as t is of the form $E[\textbf{event}\ a; t']$, and $t^{\dagger_{may}} = \textbf{true}$.
– Case $t_1 \Box t_2$: Because $(t_1 \Box t_2)^{\dagger_{may}} = {t_1}^{\dagger_{may}} \vee {t_2}^{\dagger_{may}}$, $\mathtt{L}_0 \models (D, t)^{\dagger_{may}}$ if and only if $\mathtt{L}_0 \models (D, t_i)^{\dagger_{may}}$ for some $i \in \{1, 2\}$. By the induction hypothesis, the latter is equivalent to the property that t_i is of the form $E[\textbf{event}\ a; t']$ for some $i \in \{1, 2\}$, which is equivalent to the property that t is of the form $E'[\textbf{event}\ a; t']$.

□

The following lemma says that Theorem 1 holds for recursion-free programs; this is an immediate corollary of Lemmas 1 and 2, and the strong normalization property of the simply-typed λ-calculus.

Lemma 3. *Let P be a recursion-free program. Then, $a \in \textbf{Traces}(P)$ if and only if $\mathtt{L}_0 \models \Phi_{P,may}$ for $\mathtt{L}_0 = (\{\mathtt{s}_\star\}, \emptyset, \emptyset, \mathtt{s}_\star)$.*

Proof. Since $P = (D, t_0)$ is recursion-free, there exists a finite, normalizing reduction sequence $t_0 \longrightarrow_D^* t \not\longrightarrow_D$. We show the required property by induction on the length n of this reduction sequence.

– Case $n = 0$: Since $t_0 \not\longrightarrow_D$, $a \in \textbf{Traces}(P)$ if and only if $t_0 = E[\textbf{event}\ a; t]$ for some E and t. Thus, the result follows immediately from Lemma 2.
– Case $n > 0$: In this case, $t_0 \longrightarrow_D t_1 \longrightarrow_D^* t$. By the induction hypothesis, $a \in \textbf{Traces}(D, t_1)$ if and only if $\mathtt{L}_0 \models \Phi_{(D,t_1),may}$. Thus, by the definition of the reduction semantics and Lemma 1, $a \in \textbf{Traces}(D, t_0)$ if and only if $a \in \textbf{Traces}(D, t_1)$, if and only if $\mathtt{L}_0 \models \Phi_{(D,t_1),may}$, if and only if $\mathtt{L}_0 \models \Phi_{(D,t_0),may}$.

□

To prove Theorem 1 for arbitrary programs, we use the fact that the semantics of $P^{\dagger_{may}}$ may be approximated by ${P^{(i)}}^{\dagger_{may}}$, where $P^{(i)}$ is the recursion-free program obtained by unfolding recursion functions i times (a more formal definition will be given later). To guarantee the correctness of this finite approximation, we need to introduce a slightly non-standard notion of (ω-)continuous functions below.

Definition 11. *For an LTS $\mathtt{L} = (U, A, \longrightarrow, \mathtt{s_{init}})$ and a type σ, the set of* continuous elements *$Cont_{\mathtt{L},\sigma} \subseteq \mathcal{D}_{\mathtt{L},\sigma}$ and the equivalence relation $=_{cont,\mathtt{L},\sigma} \subseteq Cont_{\mathtt{L},\sigma} \times Cont_{\mathtt{L},\sigma}$ are defined by induction on σ as follows.*

$$
\begin{array}{l}
Cont_{\mathtt{L},\bullet} = \mathcal{D}_{\mathtt{L},\bullet} \qquad Cont_{\mathtt{L},\mathtt{int}} = \mathcal{D}_{\mathtt{L},\mathtt{int}} \\
Cont_{\mathtt{L},\sigma\to\tau} = \{f \in \mathcal{D}_{\mathtt{L},\sigma\to\tau} \mid \forall x_1, x_2 \in Cont_{\mathtt{L},\sigma}.(x_1 =_{cont,\mathtt{L},\sigma} x_2 \Rightarrow f(x_1) =_{cont,\mathtt{L},\tau} f(x_2)) \\
\qquad\qquad \wedge \forall \{y_i\}_{i\in\omega} \in {Cont_{\mathtt{L},\sigma}}^{(\uparrow\omega)}.f(\bigsqcup_{i\in\omega} y_i) =_{cont,\mathtt{L},\tau} \bigsqcup_{i\in\omega} f(y_i)\}. \\
=_{cont,\mathtt{L},\bullet} = \{(x, x) \mid x \in Cont_{\mathtt{L},\bullet}\} \qquad =_{cont,\mathtt{L},\mathtt{int}} = \{(x, x) \mid x \in Cont_{\mathtt{L},\mathtt{int}}\} \\
=_{cont,\mathtt{L},\sigma\to\tau} = \{(f_1, f_2) \mid f_1, f_2 \in Cont_{\mathtt{L},\sigma\to\tau} \\
\qquad\qquad \wedge \forall x_1, x_2 \in Cont_{\mathtt{L},\sigma}.(x_1 =_{cont,\mathtt{L},\sigma} x_2 \Rightarrow f_1(x_1) =_{cont,\mathtt{L},\tau} f_2(x_2)\}.
\end{array}
$$

Here, ${Cont_{\mathtt{L},\sigma}}^{(\uparrow\omega)}$ denotes the set of increasing infinite sequences $a_0 \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq \cdots$ consisting of elements of $Cont_{\mathtt{L},\sigma}$. We just write $=_{cont}$ for $=_{cont,\mathtt{L},\sigma}$ when $\mathtt{L}$ and σ are clear from the context.

Remark 2. Note that we require that a continuous function returns a continuous element only if its argument is. To see the need for this requirement, consider an LTS with a singleton state set $\{\mathtt{s}\}$, and the function: $f = \lambda x : ((\mathtt{int} \to \bullet) \to \bullet).\lambda y : (\mathtt{int} \to \bullet).x\, y$. One may expect that f is continuous in the usual sense (i.e., f preserves the limit), but for the function g defined by

$$g(p) = \begin{cases} \{\mathtt{s}\} \text{ if } \mathtt{s} \in p(n) \text{ for every } n \geq 0 \\ \emptyset \quad \text{otherwise} \end{cases},$$

$f(g)$ is *not* continuous. In fact, let p_i be $\{(n, \{\mathtt{s}\}) \mid 0 \leq n \leq i\} \cup \{(n, \emptyset) \mid n < 0 \vee n > i\}$. Then $f(g)(p_i) = \emptyset$ for every i but $f(g)(\sqcup_{i \in \omega} p_i) = \{\mathtt{s}\}$. The (non-continuous) function g above can be expressed by $(\nu X.\lambda n.\lambda p.(p(n) \wedge (X\,(n+1)\,p)))0$. □

Lemma 4. *If* $\{x_i\}_{i\in\omega}, \{y_i\}_{i\in\omega} \in \mathit{Cont}_{\mathrm{L},\sigma}{}^{(\uparrow\omega)}$ *and* $x_i =_{\mathit{cont},\mathrm{L},\sigma} y_i$ *for each* $i \in \omega$*, then* $\bigsqcup_{i\in\omega} x_i =_{\mathit{cont},\mathrm{L},\sigma} \bigsqcup_{i\in\omega} y_i$.

Proof. The proof proceeds by induction on σ. The base case, where $\sigma = \bullet$ or $\sigma = \mathtt{int}$, is trivial, as $\mathit{Cont}_{\mathrm{L},\sigma} = \mathcal{D}_{\mathrm{L},\sigma}$ and $=_{\mathit{cont},\mathrm{L},\sigma}$ is the identity relation. Let us consider the induction step, where $\sigma = \sigma_1 \to \tau$. We first check that $\bigsqcup_{i\in\omega} x_i \in \mathit{Cont}_{\mathrm{L},\sigma}$. To this end, suppose $z_1 =_{\mathit{cont},\mathrm{L},\sigma_1} z_2$. By the continuity of x_i and the assumption $z_1 =_{\mathit{cont},\mathrm{L},\sigma_1} z_2$, we have $x_i z_1 =_{\mathit{cont},\mathrm{L},\tau} x_i z_2$ for each i. By the induction hypothesis, we have $\bigsqcup_{i\in\omega}(x_i z_1) =_{\mathit{cont},\mathrm{L},\tau} \bigsqcup_{i\in\omega}(x_i z_2)$. Therefore, we have:

$$(\bigsqcup_{i\in\omega} x_i) z_1 = \bigsqcup_{i\in\omega}(x_i z_1) =_{\mathit{cont},\mathrm{L},\tau} \bigsqcup_{i\in\omega}(x_i z_2) = (\bigsqcup_{i\in\omega} x_i) z_2$$

as required. To check the second condition for $\bigsqcup_{i\in\omega} x_i \in \mathit{Cont}_{\mathrm{L},\sigma}$, suppose that $\{z_i\}_{i\in\omega} \in \mathit{Cont}_{\mathrm{L},\sigma_1}{}^{(\uparrow\omega)}$. We need to show $(\bigsqcup_{i\in\omega} x_i)(\bigsqcup_{j\in\omega} z_j) =_{\mathit{cont},\mathrm{L},\tau} \bigsqcup_{j\in\omega}((\bigsqcup_{i\in\omega} x_i) z_j)$. By the continuity of x_i and the induction hypothesis, we have indeed:

$$\begin{aligned}&(\textstyle\bigsqcup_{i\in\omega} x_i)(\bigsqcup_{j\in\omega} z_j) = \bigsqcup_{i\in\omega}(x_i(\bigsqcup_{j\in\omega} z_j))\\ &=_{\mathit{cont},\mathrm{L},\tau} \textstyle\bigsqcup_{i\in\omega}(\bigsqcup_{j\in\omega}(x_i z_j)) = \bigsqcup_{j\in\omega}(\bigsqcup_{i\in\omega}(x_i z_j)) = \bigsqcup_{j\in\omega}((\bigsqcup_{i\in\omega} x_i) z_j).\end{aligned}$$

Thus, we have proved $\bigsqcup_{i\in\omega} x_i \in \mathit{Cont}_{\mathrm{L},\sigma}$. The proof of $\bigsqcup_{i\in\omega} y_i \in \mathit{Cont}_{\mathrm{L},\sigma}$ is the same.

It remains to check that $z =_{\mathit{cont},\mathrm{L},\sigma_1} w$ implies $(\bigsqcup_{i\in\omega} x_i) z =_{\mathit{cont},\mathrm{L},\tau} (\bigsqcup_{i\in\omega} y_i) w$. Suppose $z =_{\mathit{cont},\mathrm{L},\sigma_1} w$. Then we have:

$$(\textstyle\bigsqcup_{i\in\omega} x_i) z = \bigsqcup_{i\in\omega}(x_i z) =_{\mathit{cont},\mathrm{L},\tau} \bigsqcup_{i\in\omega}(y_i w) = (\bigsqcup_{i\in\omega} y_i) w.$$

as required. Note that $x_i z =_{\mathit{cont},\mathrm{L},\tau} y_i w$ follows from the assumptions $x_i =_{\mathit{cont},\mathrm{L},\sigma} y_i$ and $z =_{\mathit{cont},\mathrm{L},\sigma_1} w$, and then we have applied the induction hypothesis to obtain $\bigsqcup_{i\in\omega}(x_i z) =_{\mathit{cont},\mathrm{L},\tau} \bigsqcup_{i\in\omega}(y_i w)$. This completes the proof for the induction step. □

The following lemma guarantees the continuity of the functions expressed by fixpoint-free $\mathrm{HFL}_{\mathbf{Z}}$ formulas.

Lemma 5 (continuity of fixpoint-free unctions). *Let* $\mathtt{L}$ *be an LTS. If* φ *is a closed, fixpoint-free* $\mathrm{HFL}_{\mathbf{Z}}$ *formula of type* τ*, then* $[\![\varphi]\!] \in \mathit{Cont}_{\mathrm{L},\tau}$.

Proof. We write $[\![\Delta]\!]_{cont}$ for the set of valuations: $\{\rho \in [\![\Delta]\!] \mid \rho(f) \in Cont_{\mathrm{L},\sigma}$ for each $f : \sigma \in \Delta\}$, and $=_{cont,\mathrm{L},\Delta}$ for:

$$\{(\rho_1, \rho_2) \in [\![\Delta]\!]_{cont} \times [\![\Delta]\!]_{cont} \mid \rho_1(x) =_{cont,\mathrm{L},\sigma} \rho_2(x) \text{ for every } x : \sigma \in \Delta\}.$$

We show the following property by induction on the derivation of $\Delta \vdash \varphi : \sigma$:

If φ is fixpoint-free and $\Delta \vdash \varphi : \sigma$, then
(i) $\rho_1 =_{cont,\mathrm{L},\Delta} \rho_2$ imply $[\![\Delta \vdash \varphi : \sigma]\!](\rho_1) =_{cont,\mathrm{L},\sigma} [\![\Delta \vdash \varphi : \sigma]\!](\rho_2)$.
(ii) For any increasing sequence of valuations $\rho_0 \sqsubseteq \rho_1 \sqsubseteq \rho_2 \sqsubseteq \cdots$ such that $\rho_i \in [\![\Delta]\!]_{cont}$ for each $i \in \omega$, $[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i) =_{cont,\mathrm{L},\sigma} \bigsqcup_{i\in\omega}[\![\Delta \vdash \varphi : \sigma]\!](\rho_i)$.

Then, the lemma would follow as a special case, where $\Delta = \emptyset$. We perform case analysis on the last rule used for deriving $\Delta \vdash \varphi : \sigma$. We discuss only the main cases; the other cases are similar or straightforward.

– Case HT-Var: In this case, $\varphi = X$ and $\Delta = \Delta', X : \sigma$. The condition (i) follows immediately by: $[\![\Delta \vdash \varphi : \sigma]\!](\rho_1) = \rho_1(X) =_{cont,\mathrm{L},\sigma} \rho_2(X) = [\![\Delta \vdash \varphi : \sigma]\!](\rho_2)$. To see (ii), suppose $\rho_0 \sqsubseteq \rho_1 \sqsubseteq \rho_2 \sqsubseteq \cdots$, with $\rho_i \in [\![\Delta]\!]_{cont}$ for each $i \in \omega$. Then, we have

$$[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i) = (\sqcup_{i\in\omega}\rho_i)(X) = \sqcup_{i\in\omega}(\rho_i(X)) = \sqcup_{i\in\omega}([\![\Delta \vdash \varphi : \sigma]\!](\rho_i)).$$

By Lemma 4, $\sqcup_{i\in\omega}(\rho_i(X)) \in Cont_{\mathrm{L},\sigma}$. We have thus $[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i) =_{cont,\mathrm{L},\sigma} \sqcup_{i\in\omega}([\![\Delta \vdash \varphi : \sigma]\!](\rho_i))$ as required.

– Case HT-Some: In this case, $\varphi = \langle a\rangle\varphi'$ with $\Delta \vdash \varphi : \bullet$ and $\sigma = \bullet$. The condition (i) is trivial since $Cont_{\mathrm{L},\bullet} = \mathcal{D}_{\mathrm{L},\bullet}$. We also have the condition (ii) by:

$$\begin{aligned}
&[\![\Delta \vdash \langle a\rangle\varphi' : \sigma]\!](\sqcup_{i\in\omega}\rho_i) = \{\mathsf{s} \mid \exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\sqcup_{i\in\omega}\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\}\\
&= \{\mathsf{s} \mid \exists \mathsf{s}' \in \sqcup_{i\in\omega}([\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_i)).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\} \qquad \text{(by induction hypothesis)}\\
&= \sqcup_{i\in\omega}\{\mathsf{s} \mid \exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\} = \sqcup_{i\in\omega}[\![\Delta \vdash_{\mathtt{H}} \langle a\rangle\varphi' : \bullet]\!](\rho_i).
\end{aligned}$$

– Case HT-All: In this case, $\varphi = [a]\varphi'$ with $\Delta \vdash \varphi : \bullet$ and $\sigma = \bullet$. The condition (i) is trivial. The condition (ii) follows by:

$$\begin{aligned}
&[\![\Delta \vdash [a]\varphi' : \sigma]\!](\sqcup_{i\in\omega}\rho_i) = \{\mathsf{s} \mid \forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\sqcup_{i\in\omega}\rho_i)\}\\
&= \{\mathsf{s} \mid \forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in \sqcup_{i\in\omega}([\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_i))\}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by induction hypothesis)}\\
&= \sqcup_{i\in\omega}\{\mathsf{s} \mid \forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_i)\} \qquad (*)\\
&= \sqcup_{i\in\omega}[\![\Delta \vdash_{\mathtt{H}} [a]\varphi' : \bullet]\!](\rho_i).
\end{aligned}$$

To see the direction $\subseteq$ in step (*), suppose $\forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in \sqcup_{i\in\omega}([\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_i))$ holds. Since U is finite, the set $\{\mathsf{s}' \mid \mathsf{s} \xrightarrow{a} \mathsf{s}'\}$ is a finite set $\{\mathsf{s}_1, \ldots, \mathsf{s}_k\}$. For each $j \in \{1, \ldots, k\}$, there exists i_j such that $\mathsf{s}_j \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_{i_j})$. Let $i' = \max(i_1, \ldots, i_k)$. Then we have $\mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!](\rho_{i'})$ for every

$s' \in \{s_1, \ldots, s_k\}$. We have thus $\forall s' \in U.s \xrightarrow{a} s' \Rightarrow s' \in [\![\Delta \vdash_H \varphi' : \bullet]\!](\rho_{i'})$, which implies s belongs to the set in the righthand side of (*).

To see the converse (i.e., $\supseteq$), suppose $\forall s' \in U.s \xrightarrow{a} s' \Rightarrow s' \in [\![\Delta \vdash_H \varphi' : \bullet]\!](\rho_i)$ for some $i \in \omega$. Then, $\forall s' \in U.s \xrightarrow{a} s' \Rightarrow s' \in \sqcup_{i\in\omega}([\![\Delta \vdash_H \varphi' : \bullet]\!](\rho_i))$ follows immediately.

– Case HT-Abs: In this case, $\varphi = \lambda X : \sigma_1.\varphi'$, with $\Delta, X : \sigma_1 \vdash \varphi' : \tau$ and $\sigma = \sigma_1 \to \tau$. To prove the condition (i), suppose $\rho_1 =_{cont,L,\Delta} \rho_2$. Let $f_j = [\![\Delta \vdash \lambda X : \sigma_1.\varphi' : \sigma_1 \to \tau]\!](\rho_j)$ for $j \in \{1, 2\}$. We first check $f_j \in Cont_{L,\sigma_1\to\tau}$. Suppose $x_1 =_{cont,L,\sigma_1} x_2$. Then, by the induction hypothesis, we have

$$f_j\, x_1 = [\![\Delta, X{:}\sigma_1 \vdash \varphi' : \tau]\!](\rho_j\{X \mapsto x_1\}) =_{cont,L,\tau} [\![\Delta, X{:}\sigma_1 \vdash \varphi' : \tau]\!](\rho_j\{X \mapsto x_2\}) = f_j\, x_2.$$

To check the second condition for $f_j \in Cont_{L,\sigma}$, let $\{y_i\}_{i\in\omega} \in {Cont_{L,\sigma_1}}^{(\uparrow\omega)}$. Then we have:

$$\begin{aligned}
&f_j(\sqcup_{i\in\omega} y_i) = [\![\Delta, X : \sigma_1 \vdash \varphi' : \tau]\!](\rho_j\{X \mapsto \sqcup_{i\in\omega} y_i\})\\
&= [\![\Delta, X : \sigma_1 \vdash \varphi' : \tau]\!](\sqcup_{i\in\omega}(\rho_j\{X \mapsto y_i\}))\\
&=_{cont} \sqcup_{i\in\omega} [\![\Delta, X : \sigma_1 \vdash \varphi' : \tau]\!](\rho_j\{X \mapsto y_i\}) \qquad \text{(by the induction hypothesis)}\\
&= \sqcup_{i\in\omega} f_j(y_i)
\end{aligned}$$

as required.

To show $f_1 =_{cont,L,\sigma_1\to\tau} f_2$, assume again that $x_1 =_{cont,L,\sigma_1} x_2$. Then, $\rho_1\{X \mapsto x_1\} =_{cont,\Delta,X:\sigma_1} \rho_2\{X \mapsto x_2\}$. Therefore, by the induction hypothesis, we have

$$f_1(x) = [\![\Delta, X{:}\sigma_1 \vdash \varphi' : \tau]\!](\rho_1\{X \mapsto x_1\}) =_{cont,L,\tau} [\![\Delta, X{:}\sigma_1 \vdash \varphi' : \tau]\!](\rho_2\{X \mapsto x_2\}) = f_2(x).$$

This completes the proof of the condition (i).

To prove the condition (ii), suppose $\rho_0 \sqsubseteq \rho_1 \sqsubseteq \rho_2 \sqsubseteq \cdots$ with $\rho_i \in [\![\Delta]\!]_{cont}$ for each $i \in \omega$. We need to show $[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i) =_{cont,L,\sigma} \sqcup_{i\in\omega}[\![\Delta \vdash \varphi : \sigma]\!](\rho_i)$. We first check that both sides of the equality belong to $Cont_{L,\sigma}$. By Lemma 4, $\sqcup_{i\in\omega}\rho_i \in [\![\Delta]\!]_{cont}$. Thus, by the condition (i) (where we set both ρ_1 and ρ_2 to $\sqcup_{i\in\omega}\rho_i$), we have $[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i) =_{cont,L,\sigma} [\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i)$, which implies $[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i) \in Cont_{L,\sigma}$. For the righthand side, by the condition (i), we have $[\![\Delta \vdash \varphi : \sigma]\!](\rho_i) \in Cont_{L,\sigma}$ for each i. By Lemma 4, we have $\sqcup_{i\in\omega}[\![\Delta \vdash \varphi : \sigma]\!](\rho_i)$ as required.

It remains to check that $x =_{cont,L,\sigma_1} y$ implies $[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i)(x) =_{cont,L,\tau} \sqcup_{i\in\omega}[\![\Delta \vdash \varphi : \sigma]\!](\rho_i)(y)$. Suppose $x =_{cont,L,\sigma_1} y$. Then we have

$$\begin{aligned}
&[\![\Delta \vdash \varphi : \sigma]\!](\sqcup_{i\in\omega}\rho_i)(x) = [\![\Delta, X : \sigma_1 \vdash \varphi' : \sigma]\!]((\sqcup_{i\in\omega}\rho_i)\{X \mapsto x\})\\
&= [\![\Delta, X : \sigma_1 \vdash \varphi' : \sigma]\!](\sqcup_{i\in\omega}(\rho_i\{X \mapsto x\}))\\
&=_{cont} [\![\Delta, X : \sigma_1 \vdash \varphi' : \sigma]\!](\sqcup_{i\in\omega}(\rho_i\{X \mapsto y\}))\\
&\qquad\qquad \text{(by the induction hypothesis, (i))}\\
&=_{cont} \sqcup_{i\in\omega} [\![\Delta, X : \sigma_1 \vdash \varphi' : \sigma]\!](\rho_i\{X \mapsto y\})\\
&\qquad\qquad \text{(by the induction hypothesis, (ii))}\\
&= \sqcup_{i\in\omega}[\![\Delta \vdash \varphi : \sigma]\!](\rho_i)(y)
\end{aligned}$$

as required.

– Case HT-App: In this case, $\varphi = \varphi_1\varphi_2$ and $\sigma = \tau$, with $\Delta \vdash \varphi_1 : \sigma_2 \to \tau$, $\Delta \vdash \varphi_2 : \sigma_2$. The condition (i) follows immediately from the induction hypothesis. To prove the condition (ii), suppose $\rho_0 \sqsubseteq \rho_1 \sqsubseteq \rho_2 \sqsubseteq \cdots$ with $\rho_i \in [\![\Delta]\!]_{cont}$ for each $i \in \omega$. Then we have

$$
\begin{aligned}
&[\![\Delta \vdash \varphi_1\varphi_2 : \tau]\!](\sqcup_{i\in\omega}\rho_i) = [\![\Delta \vdash \varphi_1 : \sigma_2 \to \tau]\!](\sqcup_{i\in\omega}\rho_i)[\![\Delta \vdash \varphi_2 : \sigma_2]\!](\sqcup_{i\in\omega}\rho_i)\\
&=_{cont} (\sqcup_{i\in\omega}[\![\Delta \vdash \varphi_1 : \sigma_2 \to \tau]\!](\rho_i))(\sqcup_{j\in\omega}[\![\Delta \vdash \varphi_2 : \sigma_2]\!](\rho_j))\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by the induction hypothesis)}\\
&= \sqcup_{i\in\omega}([\![\Delta \vdash \varphi_1 : \sigma_2 \to \tau]\!](\rho_i)(\sqcup_{j\in\omega}[\![\Delta \vdash \varphi_2 : \sigma_2]\!](\rho_j))\\
&=_{cont} \sqcup_{i\in\omega} \sqcup_{j\in\omega}([\![\Delta \vdash \varphi_1 : \sigma_2 \to \tau]\!](\rho_i)([\![\Delta \vdash \varphi_2 : \sigma_2]\!](\rho_j))\\
&\qquad\qquad \text{(by the continuity of } [\![\Delta \vdash \varphi_1 : \sigma_2 \to \tau]\!](\rho_i) \text{ and Lemma 4)}\\
&= \sqcup_{i\in\omega}([\![\Delta \vdash \varphi_1 : \sigma_2 \to \tau]\!](\rho_i)([\![\Delta \vdash \varphi_2 : \sigma_2]\!](\rho_i))\\
&= \sqcup_{i\in\omega}([\![\Delta \vdash \varphi_1\varphi_2 : \tau]\!](\rho_i)
\end{aligned}
$$

as required.

□

The following is an immediate corollary of the lemma above (see, e.g., [49]).

Lemma 6 (fixpoint of continuous functions). *Let* L *be an LTS. If* $f \in Cont_{\mathrm{L},\tau\to\tau}$, *then* $\mathbf{lfp}_\tau(f) =_{cont,\mathrm{L},\tau} \bigsqcup_{i\in\omega} f^i(\bot_{\mathrm{L},\tau})$.

Proof. By the continuity of f, we have $f(\bigsqcup_{i\in\omega} f^i(\bot_{\mathrm{L},\tau})) =_{cont,\mathrm{L},\tau} \bigsqcup_{i\in\omega} f^i(\bot_{\mathrm{L},\tau})$. Thus, by (transfinite) induction, we have $f^\beta(\bot) =_{cont,\mathrm{L},\tau} \bigsqcup_{i\in\omega} f^i(\bot_{\mathrm{L},\tau})$. for every ordinal $\beta \geq \omega$. Since $\mathbf{lfp}_\tau(f) = f^\beta(\bot)$ for some ordinal β [49], we have the required result.

The detail of the transfinite induction is given as follows. For a given ordinal number β, let us define $f^\beta(\bot_{\mathrm{L},\tau})$ by transfinite induction. If $\beta = \beta' + 1$, then $f^\beta(\bot_{\mathrm{L},\tau}) = f(f^{\beta'}(\bot_{\mathrm{L},\tau}))$. If β is a limit ordinal, then $f^\beta(\bot_{\mathrm{L},\tau}) = \bigsqcup_{\beta'<\beta} f^{\beta'}(\bot_{\mathrm{L},\tau})$. We prove that $f^\omega(\bot_{\mathrm{L},\tau}) =_{cont} f^\beta(\bot_{\mathrm{L},\tau})$ for every $\beta > \omega$ by transfinite induction. We have proved the claim for $\beta = \omega + 1$, i.e. $f^\omega(\bot_{\mathrm{L},\tau}) =_{cont} f^{\omega+1}(\bot_{\mathrm{L},\tau})$.

– If $\beta = \beta' + 1$, by the induction hypothesis, $f^\omega(\bot_{\mathrm{L},\tau}) =_{cont} f^{\beta'}$. Since f is continuous,

$$f^\beta(\bot_{\mathrm{L},\tau}) = f(f^{\beta'}(\bot_{\mathrm{L},\tau})) =_{cont} f(f^\omega(\bot_{\mathrm{L},\tau})) =_{cont} f^\omega(\bot_{\mathrm{L},\tau}).$$

– Assume that β is a limit ordinal and $\tau = \sigma_1 \to \cdots \to \sigma_k \to \bullet$. Assume also $x_i =_{cont,\mathrm{L},\sigma_i} y_i$ for each $1 \leq i \leq k$. By the induction hypothesis, for every $\beta' < \beta$, we have

$$f^{\beta'}(\bot_{\mathrm{L},\tau}) =_{cont} f^\omega(\bot_{\mathrm{L},\tau})$$

and thus

$$f^{\beta'}(\bot_{\mathrm{L},\tau})\, x_1 \, \ldots \, x_k =_{cont} f^\omega(\bot_{\mathrm{L},\tau})\, y_1 \, \ldots \, y_k.$$

Since $=_{cont}$ on $Cont_{\mathrm{L},\bullet}$ is the standard equivalence $=$, for every $\beta' < \beta$, we have

$$f^{\beta'}(\bot_{\mathrm{L},\tau})\, x_1 \, \ldots \, x_k = f^\omega(\bot_{\mathrm{L},\tau})\, y_1 \, \ldots \, y_k.$$

Since the limit of a function is defined pointwise,

$$\left(\bigsqcup_{\beta'<\beta} f^{\beta'}(\bot_{L,\tau})\right) x_1 \dots x_k = \bigsqcup_{\beta'<\beta}(f^{\beta'}(\bot_{L,\tau})\, x_1 \dots x_k)$$
$$= \bigsqcup_{\beta'<\beta}(f^{\omega}(\bot_{L,\tau})\, y_1 \dots y_k)$$
$$= f^{\omega}(\bot_{L,\tau})\, y_1 \dots y_k.$$

Thus, we have $f^{\beta}(\bot_{L,\tau}) =_{cont} f^{\omega}$.

Since $\mathbf{lfp}_{\tau}(f) = f^{\beta}(\bot_{L,\tau})$ for some ordinal β, we have

$$\bigsqcup_{i\in\omega} f^{i}(\bot_{L,\tau}) = f^{\omega}(\bot_{L,\tau}) =_{cont} f^{\beta}(\bot_{L,\tau}) = \mathbf{lfp}_{\tau}(f)$$

as required. □

We are now ready to prove Theorem 1. Below we extend continuous functions to those on tuples, and make use of Bekić property between a simultaneous recursive definition of multiple functions and a sequence of recursive definitions; see, e.g., [51], Chapter 10.

Proof (of Theorem 1). Given a program $P = (D, t)$ with $D = \{f_1\,\widetilde{x_1} = t_1, \dots, f_n\,\widetilde{x_n} = t_n\}$, we write $P^{(i)}$ for the recursion-free program $(D^{(i)}, [f_1^{(i)}/f_1, \dots, f_n^{(i)}/f_n]t)$ where: $D^{(i)} = \{f_k^{(j+1)}\,\widetilde{x_k} = [f_1^{(j)}/f_1, \dots, f_n^{(j)}/f_n]t_k \mid j \in \{1, \dots, i-1\}, k \in \{1, \dots, n\}\} \cup \{f_k^{(0)}\,\widetilde{x_k} = (\,) \mid k \in \{1, \dots, n\}\}$. Then, we obtain the required result as follows.

$\mathrm{L}_0 \models \Phi_{P,may}$
$\Leftrightarrow \mathsf{s}_\star \in [\![t^{\dagger_{may}}]\!](\{(f_1, \dots, f_n) \mapsto \mathbf{lfp}(\lambda(f_1, \dots, f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{may}}]\!], \dots, \lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{may}}]\!]))\})$
(Bekić property)
$\Leftrightarrow \mathsf{s}_\star \in [\![t^{\dagger_{may}}]\!](\{(f_1, \dots, f_n) \mapsto$
$\bigsqcup_{i\in\omega}(\lambda(f_1, \dots, f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{may}}]\!], \dots, \lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{may}}]\!]))^i(\lambda\widetilde{x_1}.\emptyset, \dots, \lambda\widetilde{x_n}.\emptyset)\})$
(Lemmas 5 and 6)
$\Leftrightarrow \mathsf{s}_\star \in [\![t^{\dagger_{may}}]\!](\{(f_1, \dots, f_n) \mapsto$
$(\lambda(f_1, \dots, f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{may}}]\!], \dots, \lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{may}}]\!]))^i(\lambda\widetilde{x_1}.\emptyset, \dots, \lambda\widetilde{x_n}.\emptyset)\})$
for some i (Lemmas 5)
$\Leftrightarrow \mathrm{L}_0 \models \Phi_{P^{(i)},may}$ for some i (by the definition of $P^{(i)}$)
$\Leftrightarrow a \in \mathbf{Traces}(P^{(i)})$ for some i (Lemma 3)
$\Leftrightarrow a \in \mathbf{Traces}(P)$ □

B.2 Proofs for Section 5.2

We first prepare lemmas corresponding to Lemmas 1–3.

Lemma 7. *Let (D, t) be a program. If $t \longrightarrow_D t'$, then $[\![(D, t)^{\dagger_{must}}]\!] = [\![(D, t')^{\dagger_{must}}]\!]$.*

Proof. Almost the same as the proof of Lemma 1. □

Lemma 8. *Let* (D, t) *be a program and* $t \not\longrightarrow_D$. *Then,* $\mathtt{L}_0 \models (D, t)^{\dagger_{must}}$ *if and only if* t *is of the form* $C[\textbf{event}\ a; t_1, \ldots, \textbf{event}\ a; t_k]$, *where* C *is a (multi-hole) context generated by the syntax:* $C ::= [\,]_i \mid C_1 \Box C_2$.

Proof. The proof proceeds by induction on the structure of t. By the condition $t \not\longrightarrow_D$, t is generated by the following grammar:

$$t ::= (\,) \mid \textbf{event}\ a; t' \mid t_1 \Box t_2.$$

- Case $t = (\,)$: The result follows immediately, as t is not of the form $C[\textbf{event}\ a; t_1, \ldots, \textbf{event}\ a; t_k]$ and $t^{\dagger_{must}} = \textbf{false}$.
- Case $t = \textbf{event}\ a; t'$: The result follows immediately, as t is of the form $C[\textbf{event}\ a; t_1, \ldots, \textbf{event}\ a; t_k]$ (where $C = [\,]_1$ and $k = 1$) and $t^{\dagger_{must}} = \textbf{true}$.
- Case $t_1 \Box t_2$: Because $(t_1 \Box t_2)^{\dagger_{must}} = {t_1}^{\dagger_{must}} \wedge {t_2}^{\dagger_{must}}$, $\mathtt{L}_0 \models (D, t)^{\dagger_{must}}$ if and only if $\mathtt{L}_0 \models (D, t_i)^{\dagger_{must}}$ for each $i \in \{1, 2\}$. By the induction hypothesis, the latter is equivalent to the property that t_i is of the form $C_i[\textbf{event}\ a; t_{i,1}, \ldots, \textbf{event}\ a; t_{i,k_i}]$ for each $i \in \{1, 2\}$, which is equivalent to the property that t is of the form $C[\textbf{event}\ a; t_1, \ldots, \textbf{event}\ a; t_k]$ (where $C = C_1 \Box C_2$).

□

Lemma 9. *Let* P *be a recursion-free program. Then,* $\textbf{\textit{Must}}_a(P)$ *if and only if* $\mathtt{L}_0 \models \Phi_{P,must}$ *for* $\mathtt{L}_0 = (\{\mathtt{s_{init}}\}, \emptyset, \emptyset, \mathtt{s_{init}})$.

Proof. Since $P = (D, t_0)$ is recursion-free, there exists a finite, normalizing reduction sequence $t_0 \longrightarrow_D^* t \not\longrightarrow_D$. We show the required property by induction on the length n of this reduction sequence.

- Case $n = 0$: Since $t_0 \not\longrightarrow_D$, the result follows immediately from Lemma 8.
- Case $n > 0$: In this case, $t_0 \longrightarrow_D t_1 \longrightarrow_P^* t$. Thus, by the induction hypothesis, the definition of the reduction semantics and Lemma 7, $\textbf{Must}_a(D, t_0)$ if and only if $\textbf{Must}_a(D, t_1)$, if and only if $\mathtt{L}_0 \models \Phi_{(D,t_1),must}$, if and only if $\mathtt{L}_0 \models \Phi_{(D,t_0),must}$.

Proof (Theorem 2). Let $P^{(i)}$ be the the recursion-free program defined in the proof of Theorem 1. Then, we obtain the required result as follows.

$\mathtt{L}_0 \models \Phi_{P,must}$
$\Leftrightarrow \mathtt{s}_\star \in [\![t^{\dagger_{must}}]\!](\{(f_1, \ldots, f_n) \mapsto \textbf{lfp}(\lambda(f_1, \ldots, f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{must}}]\!], \ldots, \lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{must}}]\!]))\})$
(Bekić property)
$\Leftrightarrow \mathtt{s}_\star \in [\![t^{\dagger_{must}}]\!](\{(f_1, \ldots, f_n) \mapsto \bigsqcup_{i\in\omega}(\lambda(f_1, \ldots, f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{must}}]\!], \ldots, \lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{must}}]\!]))^i(\lambda\widetilde{x_1}.\emptyset, \ldots, \lambda\widetilde{x_n}.\emptyset)\})$
((Lemmas 5 and 6)
$\Leftrightarrow \mathtt{s}_\star \in [\![t^{\dagger_{must}}]\!](\{(f_1, \ldots, f_n) \mapsto (\lambda(f_1, \ldots, f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{must}}]\!], \ldots, \lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{must}}]\!]))^i(\lambda\widetilde{x_1}.\emptyset, \ldots, \lambda\widetilde{x_n}.\emptyset)\})$ for some i
$\Leftrightarrow \mathtt{L}_0 \models \Phi_{P^{(i)},must}$ for some i
(by the definition of $P^{(i)}$)
$\Leftrightarrow \textbf{Must}_a(P^{(i)})$ for some i (Lemma 9)
$\Leftrightarrow \textbf{Must}_a(P)$.

□

C Proofs for Section 6

We first modify the reduction semantics in Figure 7 by adding the following rule for distributing events with respect to $\Box$:

$$\frac{}{E[\mathbf{event}\ a; (t_1 \Box t_2)] \longrightarrow_D E[(\mathbf{event}\ a; t_1) \Box (\mathbf{event}\ a; t_2)]} \quad \text{(R-Dist)}$$

We write $\longrightarrow_{D,\mathtt{dist}}$ for this modified version of the reduction relation. We call an evaluation context *event-free* if it is generated only by the following syntax:

$$E ::= [\,] \mid E \Box t \mid t \Box E.$$

Lemma 10. *Let $P = (D, t)$ be a program such that $t \not\longrightarrow_{D,\mathtt{dist}}$. Then,* **FinTraces**$(P) \subseteq L$ *if and only if* $\mathtt{L}_L \models \Phi_{P,path}$.

Proof. Since $t \not\longrightarrow_{D,\mathtt{dist}}$, t must be of the form: $\Box_{i\in\{1,\dots,m\}}(\mathbf{event}\ a_{i,1}; \cdots \mathbf{event}\ a_{i,n_i}; (\,))$ (where $\Box_{i\in\{1,\dots,m\}} t_i$ denotes a combination of $t_1, \dots, t_m$ with $\Box$). Thus,

$$\begin{aligned}
&\mathbf{FinTraces}(P) \subseteq L \\
&\Leftrightarrow a_{i,1} \cdots a_{i,n_i} \in L \text{ for every } i \in \{1, \dots, m\} \\
&\Leftrightarrow \mathtt{L}_L \models \langle a_{i,1}\rangle \cdots \langle a_{i,n_i}\rangle \mathbf{true} \text{ for every } i \in \{1, \dots, m\} && \text{(by the construction of } \mathtt{L}_L) \\
&\Leftrightarrow \mathtt{L}_L \models \Phi_{P,path} && \text{(by the definition of } \Phi_{P,path}).
\end{aligned}$$

$\Box$

Lemma 11. *Let (D, t) be a program, and L be a prefix-closed regular language. If $t \longrightarrow_{D,\mathtt{dist}} t'$, then $[\![(D, t)^{\dagger_{path}}]\!]_{\mathtt{L}_L} = [\![(D, t')^{\dagger_{path}}]\!]_{\mathtt{L}_L}$.*

Proof. Let $D = \{f_1\, \widetilde{x}_1 = t_1, \dots, f_n\, \widetilde{x}_n = t_n\}$, and $(F_1, \dots, F_n)$ be the greatest fixpoint of

$$\lambda(X_1, \dots, X_n).([\![\lambda\widetilde{x}_1.t_1{}^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\{f_1 \mapsto X_1, \dots, f_n \mapsto X_n\}), \dots, [\![\lambda\widetilde{x}_n.t_n{}^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\{f_1 \mapsto X_1, \dots, f_n \mapsto X_n\})).$$

By the Bekić property, $[\![(D, t)^{\dagger_{path}}]\!]_{\mathtt{L}_L} = [\![t^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\{f_1 \mapsto F_1, \dots, f_n \mapsto F_n\})$. Thus, it suffices to show that $t \longrightarrow_D t'$ implies $[\![t^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\rho) = [\![t'^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\rho)$ for $\rho = \{f_1 \mapsto F_1, \dots, f_n \mapsto F_n\}$. We show it by case analysis on the rule used for deriving $t \longrightarrow_{D,\mathtt{dist}} t'$.

- Case R-Fun: In this case, $t = E[f_i\, \widetilde{t}]$ and $t' = E[[\widetilde{t}/\widetilde{x}_i]s]$ with $D(f_i) = \lambda\widetilde{x}_i.s$. Since $(F_1, \dots, F_n)$ is a fixpoint, we have:

$$\begin{aligned}
&[\![f_i\, \widetilde{t}]\!]_{\mathtt{L}_L}(\rho) = F_i([\![\widetilde{t}]\!]_{\mathtt{L}_L}(\rho)) \\
&= [\![\lambda\widetilde{x}_i.s]\!]_{\mathtt{L}_L}(\rho)([\![\widetilde{t}]\!]_{\mathtt{L}_L}(\rho)) \\
&= [\![[\widetilde{t}/\widetilde{x}_i]s]\!]_{\mathtt{L}_L}(\rho)
\end{aligned}$$

  Thus, we have $[\![t^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\rho) = [\![t'^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\rho)$ as required.

– Case R-IfT: In this case, $t = E[\mathbf{if}\ p(t'_1,\ldots,t'_k)\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]$ and $t' = E[t_1]$ with $([\![t'_1]\!]_{L_L},\ldots,[\![t'_k]\!]_{L_L}) \in [\![p]\!]$. Thus,$t^{\dagger_{path}} = (p(t'_1,\ldots,t'_k) \Rightarrow t_1{}^{\dagger_{path}}) \wedge (\neg p(t'_1,\ldots,t'_k) \Rightarrow t_2{}^{\dagger_{path}})$. Since $([\![t'_1]\!]_{L_L},\ldots,[\![t'_k]\!]_{L_L}) \in [\![p]\!]$, $([\![t'_1]\!]_{L_L},\ldots,[\![t'_k]\!]_{L_L}) \notin [\![\neg p]\!]$. Thus, $[\![(\mathbf{if}\ p(t'_1,\ldots,t'_k)\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2)^{\dagger_{path}}]\!]_{L_L}(\rho) = [\![(\mathbf{true} \Rightarrow t_1{}^{\dagger_{path}}) \wedge (\mathbf{false} \Rightarrow t_2{}^{\dagger_{path}})]\!]_{L_L}(\rho) = [\![t_1{}^{\dagger_{path}}]\!]_{L_L}(\rho)$. We have, therefore, $[\![t^{\dagger_{path}}]\!]_{L_L}(\rho) = [\![t'^{\dagger_{path}}]\!]_{L_L}(\rho)$ as required.
– Case R-IfF: Similar to the above case.
– Case R-Dist: In this case, $t = E[\mathbf{event}\ a;(t_1 \Box t_2)]$ and $t' = E[(\mathbf{event}\ a;t_1)\Box(\mathbf{event}\ a;t_2)]$. Thus, it suffices to show that $[\![(\mathbf{event}\ a;(t_1\Box t_2))^{\dagger_{path}}]\!]_{L_L}(\rho) = [\![((\mathbf{event}\ a;t_1)\Box(\mathbf{event}\ a;t_2))^{\dagger_{path}}]\!]_{L_L}(\rho)$. We have:

$$(\mathbf{event}\ a;(t_1\Box t_2))^{\dagger_{path}} = \langle a\rangle(t_1{}^{\dagger_{path}} \wedge t_2{}^{\dagger_{path}})$$
$$((\mathbf{event}\ a;t_1)\Box(\mathbf{event}\ a;t_2))^{\dagger_{path}} = (\langle a\rangle t_1{}^{\dagger_{path}}) \wedge (\langle a\rangle t_2{}^{\dagger_{path}})$$

Since L_L has at most one a-transition from each state, both formulas are equivalent, i.e., $[\![(\mathbf{event}\ a;(t_1\Box t_2))^{\dagger_{path}}]\!]_{L_L}(\rho) = [\![((\mathbf{event}\ a;t_1)\Box(\mathbf{event}\ a;t_2))^{\dagger_{path}}]\!]_{L_L}(\rho)$. □

Lemma 12. *Let P be a recursion-free program and L be a regular, prefix-closed language. Then,* $\mathbf{FinTraces}(P) \subseteq L$ *if and only if* $L_L \models \Phi_{P,path}$.

Proof. Since $P = (D, t_0)$ is recursion-free, there exists a finite, normalizing reduction sequence $t_0 \longrightarrow^*_{D,\mathtt{dist}} t \not\longrightarrow_{D,\mathtt{dist}}$. We show the required property by induction on the length n of this reduction sequence. The base case follows immediately from Lemma 10. For the induction step (where $n > 0$), we have: $t_0 \longrightarrow_D t_1 \longrightarrow^*_D t$. By the induction hypothesis, $\mathbf{FinTraces}(D, t_1) \subseteq L$ if and only if $L_L \models \Phi_{(D,t_1),path}$. Thus, by the definition of the reduction semantics and Lemma 11, $\mathbf{FinTraces}(D, t_0) \subseteq L$, if and only if $\mathbf{FinTraces}(D, t_1) \subseteq L$, if and only if $L_L \models \Phi_{(D,t_1),path}$, if and only if $L_L \models \Phi_{(D,t_0),path}$. □

To prove Theorem 3, we introduce the (slightly non-standard) notion of co-continuity, which is dual of the continuity in Definition 11.

Definition 12. *For an LTS* $\mathtt{L} = (U, A, \longrightarrow, \mathtt{s_{init}})$ *and a type* σ, *the set of* co-continuous *elements* $Cocont_{\mathtt{L},\sigma} \subseteq \mathcal{D}_{\mathtt{L},\sigma}$ *and the equivalence relation* $=_{cocont,\mathtt{L},\sigma} \subseteq Cocont_{\mathtt{L},\sigma} \times Cocont_{\mathtt{L},\sigma}$ *are defined by induction on* σ *as follows.*

$$Cocont_{\mathtt{L},\bullet} = \mathcal{D}_{\mathtt{L},\bullet}$$
$$Cocont_{\mathtt{L},\mathtt{int}} = \mathcal{D}_{\mathtt{L},\mathtt{int}}$$
$$Cocont_{\mathtt{L},\sigma\to\tau} = \{f \in \mathcal{D}_{\mathtt{L},\sigma\to\tau} \mid \forall x_1, x_2 \in Cocont_{\mathtt{L},\sigma}.(x_1 =_{cocont,\mathtt{L},\sigma} x_2 \Rightarrow f(x_1) =_{cocont,\mathtt{L},\tau} f(x_2)) \wedge \forall \{y_i\}_{i\in\omega} \in Cocont_{\mathtt{L},\sigma}{}^{(\downarrow\omega)}.f(\textstyle\bigsqcap_{i\in\omega} y_i) =_{cocont,\mathtt{L},\tau} \bigsqcap_{i\in\omega} f(y_i)\}.$$
$$=_{cocont,\mathtt{L},\bullet} = \{(x,x) \mid x \in Cocont_{\mathtt{L},\bullet}\}$$
$$=_{cocont,\mathtt{L},\mathtt{int}} = \{(x,x) \mid x \in Cocont_{\mathtt{L},\mathtt{int}}\}$$
$$=_{cocont,\mathtt{L},\sigma\to\tau} = \{(f_1,f_2) \mid f_1, f_2 \in Cocont_{\mathtt{L},\sigma\to\tau} \wedge \forall x_1,x_2 \in Cocont_{\mathtt{L},\sigma}.x_1 =_{cocont,\mathtt{L},\sigma} x_2 \Rightarrow f_1(x_1) =_{cocont,\mathtt{L},\tau} f_2(x_2)\}.$$

Here, $Cocont_{\mathtt{L},\sigma}{}^{(\downarrow\omega)}$ *denotes the set of decreasing infinite sequences* $a_0 \sqsupseteq a_1 \sqsupseteq a_2 \sqsupseteq \cdots$ *consisting of elements of* $Cocont_{\mathtt{L},\sigma}$. *We just write* $=_{cocont}$ *for* $=_{cocont,\mathtt{L},\sigma}$ *when* $\mathtt{L}$ *and* σ *are clear from the context.*

The following lemma is analogous to Lemma 5.

Lemma 13 (cocontinuity of fixpoint-free functions). *Let* $\mathtt{L}$ *be an LTS. If* φ *is a closed, fixpoint-free HFL formula of type* τ*, then* $[\![\varphi]\!]_{\mathtt{L}} \in \mathit{Cocont}_{\mathtt{L},\tau}$.

Proof. The proof is almost the same as that of Lemma 5. We write $[\![\Delta]\!]_{cocont}$ for the set of valuations: $\{\rho \in [\![\Delta]\!]_{\mathtt{L}} \mid \rho(f) \in \mathit{Cocont}_{\mathtt{L},\sigma} \text{ for each } f : \sigma \in \Delta\}$. We show the following property by induction on the derivation of $\Delta \vdash \varphi : \sigma$:

If φ is fixpoint-free and $\Delta \vdash \varphi : \sigma$, then

(i) $[\![\Delta \vdash \varphi : \sigma]\!]_{\mathtt{L}}(\rho) \in \mathit{Cocont}_{\mathtt{L},\sigma}$ for every $\rho \in [\![\Delta]\!]_{cocont}$; and

(ii) For any decreasing sequence of interpretations $\rho_0 \sqsupseteq \rho_1 \sqsupseteq \rho_2 \sqsupseteq \cdots$ such that $\rho_i \in [\![\Delta]\!]_{cont}$ for each $i \in \omega$, $[\![\Delta \vdash \varphi : \sigma]\!]_{\mathtt{L}}(\bigsqcap_{i\in\omega} \rho_i) =_{cocont} \bigsqcap_{i\in\omega}[\![\Delta \vdash \varphi : \sigma]\!]_{\mathtt{L}}(\rho_i)$.

Then, the lemma will follow as a special case, where $\Delta = \emptyset$. We perform case analysis on the last rule used for deriving $\Delta \vdash \varphi : \sigma$. We discuss only two cases below, as the proof is almost the same as the corresponding proof for Lemma 5.

- Case HT-Some: In this case, $\varphi = \langle a \rangle \varphi'$ with $\Delta \vdash \varphi : \bullet$ and $\sigma = \bullet$. The condition (i) is trivial since $\mathit{Cocont}_{\mathtt{L},\bullet} = \mathcal{D}_{\mathtt{L},\bullet}$. We also have the condition (ii) by:

  $$\begin{aligned}
  &[\![\Delta \vdash \langle a \rangle \varphi' : \sigma]\!]_{\mathtt{L}}(\textstyle\bigsqcap_{i\in\omega} \rho_i) = \{\mathsf{s} \mid \exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\bigsqcap_{i\in\omega} \rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\} \\
  &= \{\mathsf{s} \mid \exists \mathsf{s}' \in \textstyle\bigsqcap_{i\in\omega}([\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i)).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\} \qquad \text{(by induction hypothesis)} \\
  &= \textstyle\bigsqcap_{i\in\omega}\{\mathsf{s} \mid \exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\} \qquad (*) \\
  &= \textstyle\bigsqcap_{i\in\omega}[\![\Delta \vdash_{\mathtt{H}} \langle a \rangle \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).
  \end{aligned}$$

  The step (*) is obtained as follows. Suppose $\exists \mathsf{s}' \in \bigsqcap_{i\in\omega}.[\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'$. Since $\bigsqcap_{i\in\omega}.[\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i) \subseteq [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i)$ for every i, we have $\exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'$ for every i; hence we have

  $$\mathsf{s} \in \textstyle\bigsqcap_{i\in\omega}\{\mathsf{s} \mid \exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\}.$$

  Conversely, suppose

  $$\mathsf{s} \in \textstyle\bigsqcap_{i\in\omega}\{\mathsf{s} \mid \exists \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}'\},$$

  i.e., for every i, $S_i = \{\mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i) \mid \mathsf{s} \xrightarrow{a} \mathsf{s}'\}$ must be non-empty. Since the set $\{\mathsf{s}' \mid \mathsf{s} \xrightarrow{a} \mathsf{s}'\}$ is finite, and S_i decreases monotonically, $\bigcap_{i\in\omega} S_i$ must be non-empty. Thus, we have

  $$\exists \mathsf{s}' \in \textstyle\bigsqcap_{i\in\omega}[\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).\ \mathsf{s} \xrightarrow{a} \mathsf{s}',$$

  as required.

– Case HT-All: In this case, $\varphi = [a]\varphi'$ with $\Delta \vdash \varphi : \bullet$ and $\sigma = \bullet$. The condition (i) is trivial. The condition (ii) follows by:

$$\begin{aligned}
&[\![\Delta \vdash [a]\varphi' : \sigma]\!]_{\mathtt{L}}(\textstyle\bigsqcap_{i\in\omega} \rho_i) = \{\mathsf{s} \mid \forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\textstyle\bigsqcap_{i\in\omega} \rho_i)\} \\
&= \{\mathsf{s} \mid \forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in \textstyle\bigsqcap_{i\in\omega}([\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i))\} \\
&\hfill \text{(by induction hypothesis)} \\
&= \textstyle\bigsqcap_{i\in\omega}\{\mathsf{s} \mid \forall \mathsf{s}' \in U.\mathsf{s} \xrightarrow{a} \mathsf{s}' \Rightarrow \mathsf{s}' \in [\![\Delta \vdash_{\mathtt{H}} \varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i)\} \\
&= \textstyle\bigsqcap_{i\in\omega}[\![\Delta \vdash_{\mathtt{H}} [a]\varphi' : \bullet]\!]_{\mathtt{L}}(\rho_i).
\end{aligned}$$

□

The following lemma states a standard property of cocontinuous functions [49], which can be proved in the same manner as Lemma 6.

Lemma 14 (fixpoint of cocontinuous functions). *Let* $\mathtt{L}$ *be an LTS. If* $f \in \mathit{Cocont}_{\mathtt{L},\tau\to\tau}$*, then* $\mathbf{gfp}_\tau(f) =_{\mathit{cocont},\mathtt{L},\tau} \bigsqcap_{i\in\omega} f^i(\top_{\mathtt{L},\tau})$.

Proof (Proof of Theorem 3). The result follows by:

$$\begin{aligned}
&\mathtt{L}_L \not\models \Phi_{P,path} \\
&\Leftrightarrow q_0 \notin [\![t^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\{(f_1,\dots,f_n) \mapsto \mathbf{gfp}(\lambda(f_1,\dots,f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{path}}]\!]_{\mathtt{L}_L},\dots,\lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{path}}]\!]_{\mathtt{L}_L}))\}) \\
&\hfill \text{(Bekić property)} \\
&\Leftrightarrow q_0 \notin [\![t^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\{(f_1,\dots,f_n) \mapsto \\
&\qquad \textstyle\bigsqcap_{i\in\omega}(\lambda(f_1,\dots,f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{path}}]\!]_{\mathtt{L}_L},\dots,\lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{path}}]\!]_{\mathtt{L}_L}))^i(\lambda\widetilde{x_1}.Q,\dots,\lambda\widetilde{x_n}.Q)\}) \\
&\hfill \text{(Lemmas 13 and 14)} \\
&\Leftrightarrow q_0 \notin [\![t^{\dagger_{path}}]\!]_{\mathtt{L}_L}(\{(f_1,\dots,f_n) \mapsto \\
&\qquad (\lambda(f_1,\dots,f_n).(\lambda\widetilde{x_1}.[\![{t_1}^{\dagger_{path}}]\!]_{\mathtt{L}_L},\dots,\lambda\widetilde{x_n}.[\![{t_n}^{\dagger_{path}}]\!]_{\mathtt{L}_L}))^i(\lambda\widetilde{x_1}.Q,\dots,\lambda\widetilde{x_n}.Q)\}) \\
&\qquad\quad \text{for some } i \hfill \text{(Lemma 13)} \\
&\Leftrightarrow \mathtt{L}_L \not\models \Phi_{P^{(i)},path} \text{ for some } i \\
&\Leftrightarrow \mathbf{Traces}(P^{(i)}) \not\subseteq L \text{ for some } i \hfill \text{(Lemma 12)} \\
&\Leftrightarrow \mathbf{Traces}(P) \not\subseteq L
\end{aligned}$$

□

D Game-based characterization of HES

Fix an LTS $\mathtt{L}$ and let $\mathcal{E} = (X_1^{\tau_1} =_{\alpha_1} \varphi_1; \cdots; X_n^{\tau_n} =_{\alpha_n} \varphi_n)$. The goal of this section is to construct a parity game $\mathcal{G}_{\mathtt{L},\mathcal{E}}$ characterizing the semantic interpretation of the HES $\mathcal{E}$ (or equivalently $(\mathcal{E}, X_1)$) over the LTS $\mathtt{L}$. The game-based characterization will be used to prove some results in Section 7.

D.1 Preliminary: Parity Game

A *parity game* $\mathcal{G}$ is a tuple (V_P, V_O, E, Ω) where

- V_P and V_O are disjoint sets of *Proponent and Opponent nodes*, respectively,
- $E \subseteq (V_P \cup V_O) \times (V_P \cup V_O)$ is the set of edges, and
- $\Omega : (V_P \cup V_O) \to \mathbb{N}$ is a *priority function* whose image is bounded.

We write V for $V_P \cup V_O$.

A *play* of a parity game is a (finite or infinite) sequence $v_1 v_2 \dots$ of nodes in V such that $(v_i, v_{i+1}) \in E$ for every i. We write $\cdot$ for the concatenation operation. An infinite play $v_1 v_2 \dots$ is said to satisfy the *parity condition* if $\max \mathbf{Inf}(\Omega(v_1)\Omega(v_2)\dots)$ is even, where $\mathbf{Inf}(m_1 m_2 \cdots)$ is the set of numbers that occur infinitely often in $m_1 m_2 \cdots$. A play is *maximal* if either

- it is a finite sequence $v_1 \cdots v_n$ and the last node v_n has no successor (i.e., $\{v \mid (v_n, v) \in E\} = \emptyset$), or
- it is infinite.

A maximal play is *P-winning* (or simply *winning*) if either

- it is finite and the last node is of Opponent, or
- it is infinite and satisfies the parity condition.

Let $\mathcal{S} : V^* V_P \rightharpoonup V$ be a partial function that respects E in the sense that $(v_k, \mathcal{S}(v_1 \dots v_k)) \in E$ (if $\mathcal{S}(v_1 \dots v_k)$ is defined). A play $v_1 v_2 \dots v_k$ is said to *conform with* $\mathcal{S}$ if, for every $1 \le i < k$ with $v_i \in V_P$, $\mathcal{S}(v_1 \dots v_i)$ is defined and $\mathcal{S}(v_1 \dots v_i) = v_{i+1}$. An infinite play conforms with $\mathcal{S}$ if so does every finite prefix of the play. The partial function $\mathcal{S}$ is a *P-strategy on* $V_0 \subseteq V$ (or simply a *strategy on* V_0) if it is defined on every play that conforms with $\mathcal{S}$, starts from a node in V_0 and ends with a node in V_P. A strategy is *P-winning on* $V_0 \subseteq V$ (or simply *winning on* V_0) if every maximal play that conforms with the strategy and starts from V_0 is P-winning. We say that *Proponent wins the game* $\mathcal{G}$ *on* $V_0 \subseteq V$ if there exists a P-winning strategy of $\mathcal{G}$ on V_0. An *O-strategy* and an *O-winning strategy* is defined similarly.

A strategy $\mathcal{S}$ is *memoryless* if $\mathcal{S}(v_1 \dots v_k) = f(v_k)$ for some $f : V_P \to V$.

We shall consider only games of limited shape, which we call *bipartite games*. A game is *bipartite* if $E \subseteq (V_P \times V_O) \cup (V_O \times V_P)$ and $\Omega(v_O) = 0$ for every $v_O \in V_O$.

Parity progress measure [46] is a useful tool to show that a strategy is winning. We give a modified version of the definition, applicable only to bipartite games.

Definition 13 (Parity progress measure). *Let γ be an ordinal number. Given $(\beta_1, \dots, \beta_n), (\beta'_1, \dots, \beta'_n) \in \gamma^n$, we write $(\beta_1, \dots, \beta_n) \ge_j (\beta'_1, \dots, \beta'_n)$ if $(\beta_1, \dots, \beta_{n-j+1}) \ge (\beta'_1, \dots, \beta'_{n-j+1})$ by the lexicographic ordering. The strict inequality $>_j$ ($j = 1, \dots, n$) is defined by an analogous way.*

Let $\mathcal{G}$ be a bipartite parity game and $\mathcal{S}$ be a strategy of $\mathcal{G}$ on V_0. Let n be the maximum priority in $\mathcal{G}$. A (partial) mapping $\varpi : V_O \rightharpoonup \gamma^n$ is a parity progress measure of $\mathcal{S}$ on V_0 *if it satisfies the following condition:*

> *For every finite play $\tilde{v} \cdot v_O \cdot v_P \cdot v'_O$ ($v_O, v'_O \in V_O$ and $v_P \in V_P$) that starts from V_0 and conforms with the strategy $\mathcal{S}$, both $\varpi(v_O)$ and $\varpi(v'_O)$ are defined and $\varpi(v_O) \ge_{\Omega(v_P)} \varpi(v'_O)$. Furthermore if $\Omega(v_P)$ is odd, then $\varpi(v_O) >_{\Omega(v_P)} \varpi(v'_O)$.*

Lemma 15 ([46]). *Let $\mathcal{G}$ be a bipartite parity game and $\mathcal{S}$ be a strategy of $\mathcal{G}$ on $V_0 \subseteq V$. If there exists a parity progress measure of $\mathcal{S}$ on V_0, then $\mathcal{S}$ is a winning strategy on V_0.*

Proof. Let ϖ be a parity progress measure of $\mathcal{S}$ on V_0. We prove by contradiction.

Assume an infinite play $v_1 v_2 \dots$ that conforms $\mathcal{S}$, starts from V_0 and violates the parity condition. Assume that $v_1 \in V_O$; the other case can be proved similarly. Since $\mathcal{G}$ is bipartite, $v_i \in V_O$ if and only if i is odd. Furthermore $\Omega(v_i) = 0$ for every odd i.

Let $\ell = \max \mathbf{Inf}(\Omega(v_1)\Omega(v_2)\dots)$. Then there exists an even number k such that, for every even number $i \geq k$, we have $\Omega(v_i) \leq \ell$. By definition, $\varpi(v_{i-1}) \geq_{\Omega(v_i)} \varpi(v_{i+1})$ for every even i. Since $j_1 \leq j_2$ implies $(\geq_{j_1}) \subseteq (\geq_{j_2})$, we have $\varpi(v_{i-1}) \geq_\ell \varpi(v_{i+1})$ for every even $i \geq k$. Hence we have an infinite decreasing chain $\varpi(v_{k-1}) \geq_\ell \varpi(v_{k+1}) \geq_\ell \varpi(v_{k+3}) \geq_\ell \cdots$. Furthermore this inequality is strict if $\Omega(v_i) = \ell$ because ℓ is odd. Since $\{i \mid i \geq k, \Omega(v_i) = \ell\}$ is infinite, we have an infinite strictly decreasing chain, a contradiction. □

D.2 Preliminary: Complete-Prime Algebraic Lattice

Definition 14. *Let $(A, \leq)$ be a complete lattice. For $U \subseteq A$, we write $\bigvee U$ for the least upper bound of U in A. An element $p \in A$ is a* complete prime *if (1) $p \neq \bot$, and (2) $p \leq \bigvee U$ implies $p \leq x$ for some $x \in U$ (for every $U \subseteq A$). A complete lattice is* complete-prime algebraic *[52] if $x = \bigvee\{\, p \leq x \mid p : \text{complete prime} \,\}$ for every x.*

The following is a basic property about complete-prime algebraic lattices, which will be used later.

Lemma 16. *Let $(A, \leq)$ be a complete prime algebraic lattice, and $x, y \in A$. Then the followings are equivalent.*

(i) $x \leq y$.
(ii) $p \leq x$ *implies* $p \leq y$ *for every complete prime* p.
(iii) $p \nleq y$ *implies* $p \nleq x$ *for every complete prime* p.

Proof. (i) implies (ii): Trivial from the fact $p \leq x \leq y$ implies $p \leq y$. (ii) implies (i): Assume (ii). Then, $\{p \mid p \leq x, p\text{: complete prime}\} \subseteq \{p \mid p \leq y, p\text{: complete prime}\}$. Since A is complete-prime algebraic, $x = \bigvee\{p \mid p \leq x, p\text{: complete prime}\} \leq \bigvee\{p \mid p \leq y, p\text{: complete prime}\} = y$. (iii) is just a contraposition of (ii). □

Let $(A, \leq)$ be a complete lattice and $f : A \to A$ be a monotone function. We write $\bot_A$ for the least element of A. Then A has the least fixed point of f, which can be computed by iterative application of f to $\bot_A$ as follows. Let γ be an ordinal number greater than the cardinality of A. We define a family $\{a_i\}_{i<\gamma}$ of elements in A by:

$$a_0 := \bot_A \qquad a_{\beta+1} := f(a_\beta) \qquad a_\beta := \bigvee_{\beta' < \beta} a_{\beta'} \quad \text{(if } \beta \text{ is a limit ordinal).}$$

Then a_γ is the least fixed point of f. We shall write a_β as $f^\beta(\bot_A)$.

Lemma 17. *Let $(A, \leq)$ be a complete lattice and $f : A \to A$ be a monotone function. For every complete prime $p \in A$, the minimum ordinal β such that $p \leq f^\beta(\bot)$ is a successor ordinal.*

Proof. Assume that the minimum ordinal β is a limit ordinal. By definition, $p \leq \bigvee_{\beta'<\beta} f^{\beta'}(\bot)$. Since p is a complete prime, there exists $\beta' < \beta$ such that $p \leq f^{\beta'}(\bot)$. Hence β is not the minimum, a contradiction. □

Lemma 18. *Let* L *be an LTS. Then* $(\mathcal{D}_{\mathrm{L},\tau}, \sqsubseteq_{\mathrm{L},\tau})$ *is complete-prime algebraic for every* τ.

Proof. If $\tau = \bullet$, then $\mathcal{D}_{\mathrm{L},\tau} = 2^U$ is ordered by the set inclusion and the complete primes are singleton sets $\{q\}$ ($q \in U$). For function types, $\mathcal{D}_{\mathrm{L},\sigma\to\tau}$ is the set of monotone functions ordered by the pointwise ordering. Given an element $d \in \mathcal{D}_{\mathrm{L},\sigma}$ and a complete prime $p \in \mathcal{D}_{\mathrm{L},\tau}$, consider the function $f_{d,p}$ defined by

$$f_{d,p}(x) = \begin{cases} p & (\text{if } d \leq x) \\ \bot & (\text{otherwise}). \end{cases}$$

These functions are complete primes of $\mathcal{D}_{\mathrm{L},\sigma\to\tau}$. It is not difficult to see that every element is the least upper bound of a subset of complete primes. □

D.3 Parity Game for HES

Let $\mathcal{E} = (X_1^{\tau_1} =_{\alpha_1} \varphi_1; \cdots; X_n^{\tau_n} =_{\alpha_n} \varphi_n)$ be an HES (with "the main formula" X_1) and L be an LTS.

Definition 15 (Parity game for HES). *The parity game* $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ *is defined by the following data:*

$$\begin{aligned} V_P &:= \{ (p, X_i) \mid p \in \mathcal{D}_{\mathrm{L},\tau_i},\ p\text{: complete prime} \} \\ V_O &:= \{ (x_1, \dots, x_n) \mid x_1 \in \mathcal{D}_{\mathrm{L},\tau_1}, \dots, x_n \in \mathcal{D}_{\mathrm{L},\tau_n} \} \\ E &:= \{ ((p, X_i), (x_1, \dots, x_n)) \mid p \sqsubseteq [\![\varphi_i]\!]([X_1 \mapsto x_1, \dots, X_n \mapsto x_n]) \} \\ &\quad \cup \{ ((x_1, \dots, x_n), (p, X_i)) \mid p \sqsubseteq x_i,\ p\text{: complete prime} \}. \end{aligned}$$

The priority of the opponent node is 0*; the priority of node* (p, X_i) *is* $2(n-i)$ *if* $\alpha_i = \nu$ *and* $2(n-i)+1$ *if* $\alpha_i = \mu$.

The parity game defined above is analogous to the typability game for HES defined by [20]. A position $(p, X_i) \in V_P$ represents the state where Proponent tries to show that $p \sqsubseteq [\![(\mathcal{E}, X_i)]\!]$. To show $p \sqsubseteq [\![(\mathcal{E}, X_i)]\!]$, Proponent picks a valuation $\rho = [X_1 \mapsto x_1, \dots, X_n \mapsto x_n]$ such that $p \sqsubseteq [\![\varphi_i]\!](\rho)$ indeed holds. A position $(x_1, \dots, x_n) \in V_O$ represents such a valuation, and Opponent challenges Proponent's assumption that $\rho = [X_1 \mapsto x_1, \dots, X_n \mapsto x_n]$ is a valid valuation, i.e., $x_i \sqsubseteq [\![(\mathcal{E}, X_i)]\!]$ holds for each i. To this end, Opponent chooses i, picks a complete prime p such that $p \sqsubseteq x_i$, and asks why $p \sqsubseteq [\![(\mathcal{E}, X_i)]\!]$ holds, as represented by the edge $((x_1, \dots, x_n), (p, X_i))$. (Note that Lemma 18 implies that $x_i \sqsubseteq [\![(\mathcal{E}, X_i)]\!]$ if and only if $p \sqsubseteq [\![(\mathcal{E}, X_i)]\!]$ for every complete prime such that $p \sqsubseteq x_i$; therefore, it is sufficient for Opponent to consider only complete primes.) As in the typability game for HES defined by [20], a play may continue indefinitely, in which case, the winner is determined by the largest priority of (p, X_i) visited infinitely often.

The goal of this section is to show that $p \sqsubseteq [\![\mathcal{E}]\!]_{\mathrm{L}}$ if and only if Proponent wins $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on (p, X_1) (Theorem 7 given at the end of this subsection).

We start from an alternative description of the interpretation of an HES. Let us define the family $\{\vartheta_i\}_{i=1,\dots,n}$ of substitutions by induction on $n - i$ as follows:

$$\begin{aligned}
&\vartheta_n := \text{(the identity substitution)} \\
&\vartheta_i := [X_{i+1} \mapsto \alpha_{i+1} X_{i+1}.(\vartheta_{i+1}\varphi_{i+1})] \circ \vartheta_{i+1} \qquad \text{(if } i < n\text{)}.
\end{aligned}$$

The family $\{\psi_i\}_{i=1,\dots,n}$ of formulas is defined by

$$\psi_i := \alpha_i X_i.(\vartheta_i \varphi_i).$$

Then $\vartheta_i = [X_{i+1} \mapsto \psi_{i+1}] \circ \vartheta_{i+1}$. The substitution ϑ_i maps $X_{i+1}, \dots, X_n$ to closed formulas and thus $\vartheta_i \varphi_j$ ($j \leq i$) is a formula with free variables $X_1, \dots, X_i$. The formula ψ_i has free variables $X_1, \dots, X_{i-1}$. In particular, $\psi_1 = \mathit{toHFL}(\mathcal{E}, X_1)$.

Given $1 \leq i \leq j \leq n$ and $d_k \in \mathcal{D}_{\mathrm{L},\tau_k}$ ($1 \leq k \leq n$), let us define

$$\begin{aligned}
f_{i,j}(d_1, \dots, d_j) &:= [\![\vartheta_j \varphi_i]\!]([X_1 \mapsto d_1, \dots, X_j \mapsto d_j]) \\
g_i(d_1, \dots, d_{i-1}) &:= [\![\psi_i]\!]([X_1 \mapsto d_1, \dots, X_{i-1} \mapsto d_{i-1}]).
\end{aligned}$$

Lemma 19. *We have*

$$\begin{aligned}
f_{i,n}(d_1, \dots, d_n) &= [\![\varphi_i]\!]([X_1 \mapsto d_1, \dots, X_n \mapsto d_n]) \\
f_{i,j}(d_1, \dots, d_j) &= f_{i,j+1}(d_1, \dots, d_j, g_{j+1}(d_1, \dots, d_j)) \qquad \textit{(if } j < n\textit{)} \\
g_i(d_1, \dots, d_{i-1}) &= \alpha_i y. f_{i,i}(d_1, \dots, d_{i-1}, y). \\
g_1(\,) &= [\![\mathcal{E}]\!].
\end{aligned}$$

Proof. By induction on $n - i$. For each i, the claim is proved by induction on the structure of formulas φ_i. The last claim follows from the fact that $\psi_1 = \mathit{toHFL}(\mathcal{E}, X_1)$ and:

$$g_1(\,) = \alpha_1 y. f_{1,1}(y) = \alpha_1 y. [\![\vartheta_1 \varphi_1]\!]([X_1 \mapsto y]) = [\![\alpha_1 X_1. \vartheta_1 \varphi_1]\!] = [\![\psi_1]\!].$$

□

We first prove the following lemma, which implies that the game-based characterization is complete, i.e., that $p \sqsubseteq [\![\mathcal{E}]\!]_{\mathrm{L}}$ only if Proponent wins $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on (p, X_1).

Lemma 20. *Let $(d_1, \dots, d_n)$ be an opponent node. If $d_j \sqsubseteq g_j(d_1, \dots, d_{j-1})$ for every $j \in \{1, \dots, n\}$, then Proponent wins $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on $(d_1, \dots, d_n)$.*

Proof. Let $(\star)$ be the following condition on O-nodes $(d_1, \dots, d_n)$:

$d_j \sqsubseteq g_j(d_1, \dots, d_{j-1})$ for every $j \in \{1, \dots, n\}$.

Let $V_0 = \{(d_1, \dots, d_n) \in V_O \mid (d_1, \dots, d_n) \text{ satisfies } (\star)\}$.

Let us first define a strategy $\mathcal{S}$ of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on V_0. It is defined for plays of the form $\widetilde{v} \cdot (d_1, \dots, d_n) \cdot (p, X_i)$ where $(d_1, \dots, d_n) \in V_0$. The next node is determined by the last two nodes as follows.

- Case $\alpha_i = \nu$: Then the next node is

$$(d_1, \ldots, d_{i-1}, c_i, c_{i+1}, \ldots, c_n)$$

where $c_j = g_j(d_1, \ldots, d_{i-1}, c_i, c_{i+1}, \ldots, c_{j-1})$ for $j \geq i$. By the assumption $d_i \sqsubseteq g_i(d_1, \ldots, d_{i-1})$ and Lemma 19,

$$\begin{aligned}
p &\sqsubseteq d_i \\
&\sqsubseteq g_i(d_1, \ldots, d_{i-1}) \\
&= \nu y.f_{i,i}(d_1, \ldots, d_{i-1}, y) \\
&= f_{i,i}(d_1, \ldots, d_{i-1}, \nu y.f_{i,i}(d_1, \ldots, d_{i-1}, y)) \\
&= f_{i,i}(d_1, \ldots, d_{i-1}, g_i(d_1, \ldots, d_{i-1})) \\
&= f_{i,i}(d_1, \ldots, d_{i-1}, c_i) \\
&= f_{i,i+1}(d_1, \ldots, d_{i-1}, c_i, g_{i+1}(d_1, \ldots, d_{i-1}, c_i)) \\
&= f_{i,i+1}(d_1, \ldots, d_{i-1}, c_i, c_{i+1}) \\
&\vdots \\
&= f_{i,n}(d_1, \ldots, d_{i-1}, c_i, c_{i+1}, \ldots, c_n) \\
&= [\![\varphi_i]\!]([X_1 \mapsto d_1, \ldots, X_{i-1} \mapsto d_{i-1}, X_i \mapsto c_i, \ldots, X_n \mapsto c_n]).
\end{aligned}$$

- Case $\alpha_i = \mu$: Then the next node is

$$(d_1, \ldots, d_{i-1}, e, c_{i+1}, \ldots, c_n)$$

where $c_j = g_j(d_1, \ldots, d_{i-1}, e, c_{i+1}, \ldots, c_{j-1})$ for $j > i$ and e is defined as follows. By the condition ($\star$) and Lemma 19,

$$d_i \sqsubseteq g_i(d_1, \ldots, d_{i-1}) = \mu y.f_{i,i}(d_1, \ldots, d_{i-1}, y)$$

and thus

$$d_i \sqsubseteq (\lambda y.f_{i,i}(d_1, \ldots, d_{i-1}, y))^{\beta}(\bot)$$

for some ordinal β. By the definition of edges of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$, we have $p \sqsubseteq d_i$ and thus

$$p \sqsubseteq (\lambda y.f_{i,i}(d_1, \ldots, d_{i-1}, y))^{\beta}(\bot)$$

for some β. Consider the minimum ordinal among those satisfying this condition; let β_0 be this ordinal. Since p is complete prime, β_0 is a successor ordinal by Lemma 17, i.e. $\beta_0 = \beta_0' + 1$ for some β_0'. Then we define $e = (\lambda y.f_{i,i}(x_1, \ldots, x_{i-1}, y))^{\beta'}(\bot)$. Now we have

$$\begin{aligned}
p &\sqsubseteq (\lambda y.f_{i,i}(d_1, \ldots, d_{i-1}, y))(e) \\
&= f_{i,i}(d_1, \ldots, d_{i-1}, e) \\
&= f_{i,i+1}(d_1, \ldots, d_{i-1}, e, g_{i+1}(d_1, \ldots, d_{i-1}, e)) \\
&= f_{i,i+1}(d_1, \ldots, d_{i-1}, e, c_{i+1}) \\
&\vdots \\
&= f_{i,n}(d_1, \ldots, d_{i-1}, e, c_{i+1}, \ldots, c_n) \\
&= [\![\varphi_i]\!]([X_1 \mapsto d_1, \ldots, X_{i-1} \mapsto d_{i-1}, X_i \mapsto e, X_{i+1} \mapsto c_{i+1}, \ldots, X_n \mapsto c_n]).
\end{aligned}$$

It is not difficult to see that $\mathcal{S}$ is indeed a strategy on V_0.

We prove that $\mathcal{S}$ is winning by giving a parity progress measure of $\mathcal{S}$ on V_0. Let γ be an ordinal greater than the cardinality of $\mathcal{D}_{\mathrm{L},\tau_\ell}$ for every $\ell \in \{1, \ldots, n\}$. The (partial) mapping $V_O \ni (d_1, \ldots, d_n) \mapsto (\beta_1, \ldots, \beta_n) \in \gamma^n$ is defined by:

$$\beta_i := \begin{cases} \min\{\beta < \gamma \mid d_i \leq (\lambda y. f_{i,i}(d_1, \ldots, d_{i-1}, y))^\beta(\bot)\} & (\text{if } \alpha_i = \mu) \\ 0 & (\text{if } \alpha_i = \nu). \end{cases}$$

This is well-defined on V_0. It is not difficult to see that this mapping is a parity progress measure of the strategy $\mathcal{S}$. ◻

Next, we prepare lemmas used for proving that the game-based characterization is sound, i.e., that existence of a winning strategy of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on (p, X_1) implies $p \sqsubseteq [\![\mathcal{E}]\!]_{\mathrm{L}}$. We first prove this result for a specific class of strategies, which we call *stable strategies*. Then we show that every winning strategy can be transformed to a stable one.

Definition 16 (Stable strategy). *A strategy $\mathcal{S}$ of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ is* stable *if for every play*

$$\widetilde{v} \cdot (d_1, \ldots, d_n) \cdot (p, X_i) \cdot (c_1, \ldots, c_n)$$

that conforms with $\mathcal{S}$, one has $d_1 = c_1, \ldots, d_{i-1} = c_{i-1}$ and $d_i \sqsupseteq c_i$.

An important property of a stable winning strategy is as follows.

Lemma 21. *Given $d_1 \in \mathcal{D}_{\mathrm{L},\tau_1}, \ldots, d_\ell \in \mathcal{D}_{\mathrm{L},\tau_\ell}$, we write $\mathcal{G}_{\mathrm{L},\mathcal{E}}^{d_1,\ldots,d_\ell}$ for the subgame of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ consisting of nodes*

$$\{ (p, X_i) \in V_P \mid \ell < i \} \cup \{ (d_1, \ldots, d_\ell, c_{\ell+1}, \ldots, c_n) \mid c_{\ell+1} \in \mathcal{D}_{\mathrm{L},\tau_{\ell+1}}, \ldots, c_n = \mathcal{D}_{\mathrm{L},\tau_n} \}.$$

If a stable strategy $\mathcal{S}$ wins on an opponent node $(d_1, \ldots, d_n)$, then its restriction is a winning strategy of the subgame $\mathcal{G}_{\mathrm{L},\mathcal{E}}^{d_1,\ldots,d_\ell}$ on $(d_1, \ldots, d_n)$.

Proof. It suffices to show that $\mathcal{S}$ does not get stuck in the subgame $\mathcal{G}_{\mathrm{L},\mathcal{E}}^{d_1,\ldots,d_\ell}$. Since proponent nodes are restricted to (p, X_i) with $\ell < i$, by the definition of stability, $\mathcal{S}$ does not change the first ℓ components. ◻

Lemma 22. *Let $\mathcal{S}$ be a stable winning strategy of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on $V_0 \subseteq V$. For every $1 \leq j \leq n$, we have the following propositions:*

- *(P_j): For every $1 \leq i \leq j$, if $\tilde{v} \cdot (p, X_i) \cdot (d_1, \ldots, d_n)$ conforms with $\mathcal{S}$ and starts from V_0, then $p \sqsubseteq f_{i,j}(d_1, \ldots, d_j)$.*
- *(Q_j): If $\widetilde{v} \cdot (p, X_j) \cdot (d_1, \ldots, d_n)$ conforms with $\mathcal{S}$ and starts from V_0, then $p \sqsubseteq g_j(d_1, \ldots, d_{j-1})$.*

Proof. By induction on the order $P_n < Q_n < P_{n-1} < Q_{n-1} < \cdots < P_1 < Q_1$. Let $\mathcal{S}$ be a stable winning strategy of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on $V_0 \subseteq V$.

(P_n) By the definition of the game $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ and Lemma 19, we have

$$p \sqsubseteq [\![\varphi_i]\!]([X_1 \mapsto d_1, \ldots, X_n \mapsto d_n]) = f_{i,n}(d_1, \ldots, d_n).$$

(P_j with $j < n$) We first show that $d_{j+1} \sqsubseteq g_{j+1}(d_1, \ldots, d_j)$. Since $\mathcal{D}_{\mathrm{L},\tau_{j+1}}$ is complete-prime algebraic, it suffices to show $q \sqsubseteq g_{j+1}(d_1, \ldots, d_j)$ for every complete prime $q \sqsubseteq d_{j+1}$. Given a complete prime $q \sqsubseteq d_{j+1}$,

$$\tilde{v} \cdot (p, X_i) \cdot (d_1, \ldots, d_n) \cdot (q, X_{j+1})$$

is a valid play that conforms with $\mathcal{S}$. Let

$$(c_1, \ldots, c_n) = \mathcal{S}(\, \tilde{v} \cdot (p, X_i) \cdot (d_1, \ldots, d_n) \cdot (q, X_{j+1})\,).$$

Since $\mathcal{S}$ is stable, $d_1 = c_1, \ldots, d_j = c_j$. By the induction hypothesis, $q \sqsubseteq g_{i+1}(c_1, \ldots, c_j) = g_{i+1}(d_1, \ldots, d_j)$. Since $q \sqsubseteq d_{j+1}$ is an arbitrary complete prime, we conclude that $d_{j+1} \sqsubseteq g_{j+1}(d_1, \ldots, d_j)$.

Then by Lemma 19, we have

$$p \sqsubseteq f_{i,j+1}(d_1, \ldots, d_j, d_{j+1}) \sqsubseteq f_{i,j+1}(d_1, \ldots, d_j, g_{j+1}(d_1, \ldots, d_j)) = f_{i,j}(d_1, \ldots, d_j),$$

where the first inequality follows from the induction hypothesis and the second from $d_{j+1} \sqsubseteq g_{j+1}(d_1, \ldots, d_j)$ and monotonicity of $f_{i,j+1}$.

(Q_j with $\alpha_j = \nu$) For every complete prime $q \sqsubseteq d_j$, we have a play

$$\tilde{v} \cdot (p, X_i) \cdot (d_1, \ldots, d_n) \cdot (q, X_j)$$

that conforms with $\mathcal{S}$. Let

$$(c_{q,1}, \ldots, c_{q,n}) = \mathcal{S}(\, \tilde{v} \cdot (p, X_i) \cdot (d_1, \ldots, d_n) \cdot (q, X_j)\,).$$

Since $\mathcal{S}$ is stable, $d_1 = c_{q,1}, \ldots, d_{j-1} = c_{q,j-1}$ and $d_j \sqsupseteq c_{q,j}$. By the induction hypothesis and monotonicity of $f_{j,j}$, we have

$$q \sqsubseteq f_{j,j}(c_{q,1}, \ldots, c_{q,j-1}, c_{q,j}) \sqsubseteq f_{j,j}(d_1, \ldots, d_{j-1}, d_j).$$

Since the complete prime $q \sqsubseteq d_j$ is arbitrary and $\mathcal{D}_{\mathrm{L},\mathcal{E}}$ is complete-prime algebraic, we have

$$d_j = \bigsqcup_{q \sqsubseteq d_j} q \sqsubseteq f_{j,j}(d_1, \ldots, d_{j-1}, d_j).$$

Hence d_j is a prefixed point of $\lambda y. f_{j,j}(d_1, \ldots, d_{j-1}, y)$. So $d_j \sqsubseteq \nu y. f_{j,j}(d_1, \ldots, d_{j-1}, y)$. By the induction hypothesis and Lemma 19, we have

$$\begin{aligned} p &\sqsubseteq f_{j,j}(d_1, \ldots, d_{j-1}, d_j) \\ &\sqsubseteq f_{j,j}(d_1, \ldots, d_{j-1}, \nu y. f_{j,j}(d_1, \ldots, d_{j-1}, y)) \\ &= \nu y. f_{j,j}(d_1, \ldots, d_{j-1}, y) \\ &= g_j(d_1, \ldots, d_j). \end{aligned}$$

(Q_j with $\alpha_j = \mu$) Let $\tilde{v} \cdot (p, X_j) \cdot (d_1, \ldots, d_n)$ be a play that conforms with $\mathcal{S}$ and starts from V_0. Given opponent nodes $O_1, O_2 \in V_O$, we write $O_1 \succ O_2$ if there exists a play of the form

$$\widetilde{v} \cdot (p, X_j) \cdot \widetilde{v_1} \cdot O_1 \cdot \widetilde{v_2} \cdot O_2$$

that conforms with $\mathcal{S}$ such that every proponent node in $\widetilde{v}_1$ or $\widetilde{v}_2$ is of the form (q, X_j). Every play of the form

$$\tilde{v} \cdot (p, X_j) \cdot (d_1, \dots, d_n) \cdot (p_1, X_j) \cdot (c_{1,1}, \dots, c_{1,n}) \cdot (p_2, X_j) \cdot (c_{2,1}, \dots, c_{2,n}) \dots$$

that conforms with $\mathcal{S}$ eventually terminates because $\mathcal{S}$ is a winning strategy and the priority of (p_i, X_j) is odd (recall that opponent node O_i has priority 0). Hence the relation $>$ defined above is well-founded. We prove the following claim by induction on the well-founded relation $>$:

> Let $\tilde{v} \cdot (p, X_j) \cdot (d_1, \dots, d_n) \cdot \tilde{v}_1 \cdot (c_1, \dots, c_n)$ be a play that conforms with $\mathcal{S}$ and starts from V_0. Suppose that every proponent node in $\tilde{v}_1$ is of the form (q, X_j). Then $c_j \sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)$.

It suffices to show that $q \sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)$ for every complete prime $q \sqsubseteq c_j$. Consider the play

$$\tilde{v} \cdot (p, X_j) \cdot (d_1, \dots, d_n) \cdot \tilde{v}_1 \cdot (c_1, \dots, c_n) \cdot (q, X_j) \cdot (c'_1, \dots, c'_n)$$

that conforms with $\mathcal{S}$ and starts from V_0. Because $\mathcal{S}$ is stable and $\tilde{v}_1$ does not contain a node of the form (r, X_k) with $k < j$, we have $d_1 = c_1 = c'_1, \dots, d_{j-1} = c_{j-1} = c'_{j-1}$. By the induction hypothesis of the above claim, $c'_j \sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)$. By the induction hypothesis of the lemma, $q \sqsubseteq f_{j,j}(c'_1, \dots, c'_{j-1}, c'_j)$. Hence

$$\begin{aligned} q &\sqsubseteq f_{j,j}(c'_1, \dots, c'_{j-1}, c'_j) \\ &\sqsubseteq f_{j,j}(c'_1, \dots, c'_{j-1}, \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)) \\ &\sqsubseteq f_{j,j}(d_1, \dots, d_{j-1}, \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)) \\ &= \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y). \end{aligned}$$

So $q \sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)$ for every complete prime $q \sqsubseteq c_j$ and thus $c_j \sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y))$. By the same argument, we have $d_j \sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)$. This completes the proof of the above claim.

By the induction hypothesis, the above claim and Lemma 19, we have

$$\begin{aligned} p &\sqsubseteq f_{j,j}(d_1, \dots, d_{j-1}, d_j) && \text{(induction hypothesis)} \\ &\sqsubseteq f_{j,j}(d_1, \dots, d_{j-1}, \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y)) && \text{(the above claim)} \\ &\sqsubseteq \mu y.f_{j,j}(d_1, \dots, d_{j-1}, y) \\ &= g_j(d_1, \dots, d_{j-1}) && \text{(Lemma 19).} \end{aligned}$$

□

Lemma 23. *If there exists a winning strategy of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on an Opponent node $(d_1, \dots, d_n)$, there exists a stable winning strategy on a node $(d'_1, \dots, d'_n)$ with $d_1 \sqsubseteq d'_1, \dots, d_n \sqsubseteq d'_n$.*

Proof. Let $\mathcal{S}$ be a winning strategy on the node $(d_1, \dots, d_n)$. Since $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ is a parity game, we can assume without loss of generality that $\mathcal{S}$ is memoryless. Given nodes v and v' of the game and $j \in \{1, \ldots, n\}$, we say v' is *reachable from v following $\mathcal{S}$* if there exists a play $vv_1 \ldots v_\ell v'$ that conforms with $\mathcal{S}$. We say that v' is *j-reachable from v following $\mathcal{S}$* if furthermore, for every proponent node (p, X_i) in $v_1 \ldots v_k$, we have $j < i$.

For every $k = 1, \ldots, n$, let

$$d'_k = d_k \sqcup \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is reachable from } (d_1, \ldots, d_n) \text{ following } \mathcal{S} \}.$$

Obviously $d_i \sqsubseteq d'_i$.

Let $\mathcal{S}'$ be a partial function $V^* V_P \rightharpoonup V$ defined by

$$\widetilde{v} \cdot (c_1, \ldots, c_n) \cdot (q, X_j) \mapsto (c_1, \ldots, c_{j-1}, c'_j, \ldots, c'_n)$$

where

$$c'_k = \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } j\text{-reachable from } (q, X_j) \text{ following } \mathcal{S} \}.$$

We show that $\mathcal{S}'$ is indeed a strategy on $(d'_1, \ldots, d'_n)$. Let $v_0 = (d'_1, \ldots, d'_n)$ and assume that $v_0 v_1 \ldots v_\ell$ ($\ell > 0$) is a play that conforms with $\mathcal{S}'$ and ends with an opponent node $v_\ell \in V_O$ (hence $\ell > 0$ is even). Let $v_\ell = (c_1, \ldots, c_n)$. We first prove that for every $k = 1, \ldots n$,

$$c_k \sqsupseteq \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } k\text{-reachable from } v_{\ell-1} \text{ following } \mathcal{S} \}$$

by induction on $\ell > 0$. Let $v_{\ell-1} = (p, X_j)$.

- If $k \geq j$, then by the definition of $\mathcal{S}'$,

$$\begin{aligned} c_k &= \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } j\text{-reachable from } (p, X_j) \text{ following } \mathcal{S} \} \\ &\sqsupseteq \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } k\text{-reachable from } (p, X_j) \text{ following } \mathcal{S} \} \end{aligned}$$

  because the k-reachable set is a subset of the j-reachable set.
- If $k < j$, then c_k is the k-th component of $v_{\ell-2}$ by the definition of $\mathcal{S}'$.
  - If $\ell = 2$, then

$$\begin{aligned} c_k = d'_k &= d_k \sqcup \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is reachable from } (d_1, \ldots, d_n) \text{ following } \mathcal{S} \} \\ &\sqsupseteq \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } k\text{-reachable from } v_1 \text{ following } \mathcal{S} \} \end{aligned}$$

    since v_1 is reachable from $(d_1, \ldots, d_n)$ following $\mathcal{S}$.
  - Assume that $\ell > 2$. By the induction hypothesis for the subsequence $v_0 v_1 \ldots v_{\ell-2}$, we have

$$\begin{aligned} c_k &\sqsupseteq \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } k\text{-reachable from } v_{\ell-3} \text{ following } \mathcal{S} \} \\ &\sqsupseteq \bigsqcup \{ e_k \mid (e_1, \ldots, e_n) \text{ is } k\text{-reachable from } v_{\ell-1} \text{ following } \mathcal{S} \} \end{aligned}$$

    since $v_{\ell-1}$ is k-reachable from $v_{\ell-3}$ following $\mathcal{S}$.

Let $v_0 \cdot v_1 \cdot \dots \cdot v_\ell \cdot (p, X_i)$ be a play that conforms with $\mathcal{S}'$ and starts from $v_0 = (d'_1, \dots, d'_n)$. Let $(c_1, \dots, c_n) = \mathcal{S}'(v_0 \cdot v_1 \cdot \dots \cdot v_\ell \cdot (p, X_i))$ and $(e_1, \dots, e_n) = \mathcal{S}(\,(p, X_i)\,)$. Since $(e_1, \dots, e_n)$ is reachable from (p, X_i) following $\mathcal{S}$, the above claim shows that $e_m \sqsubseteq c_m$ for every $1 \le m \le n$. Since $p \sqsubseteq [\![\varphi_i]\!]([X_1 \mapsto e_1, \dots, X_n \mapsto e_n])$, we have $p \sqsubseteq [\![\varphi_i]\!]([X_1 \mapsto c_1, \dots, X_n \mapsto c_n])$. Hence $\mathcal{S}'$ is indeed a strategy.

By definition and the above claim, it is easy to see that $\mathcal{S}'$ is a stable strategy.

We show that the strategy $\mathcal{S}'$ is winning on $(d'_1, \dots, d'_n)$. Assume that $v_0 v_1 \dots$ be an infinite play that conforms with $\mathcal{S}'$ and starts from $(d'_1, \dots, d'_n)$. Let j be the minimum index of variables appearing infinitely often in the play. Then the play can be split into

$$\widetilde{w}_0 \cdot (p_1, X_j) \cdot \widetilde{w}_1 \cdot (c_{1,1}, \dots, c_{1,n}) \cdot (p_2, X_j) \cdot \widetilde{w}_2 \cdot (c_{2,1}, \dots, c_{2,n}) \cdot (p_3, X_j) \cdot \dots$$

where $\tilde{w}_k$ is a (possible empty) sequence of nodes, which furthermore consists of $V_O \cup \{\, (q, X_\ell) \mid j < \ell \,\}$ if $k \ge 1$. For every $m = 1, 2, \dots$, we have

$$c_{m,j} = \bigsqcup \{\, e_k \mid (e_1, \dots, e_n) \text{ is } j\text{-reachable from } (p_m, X_j) \text{ following } \mathcal{S} \,\}$$

since j-th component of opponent nodes is unchanged during $\widetilde{w}_m$. By the definition of edges of the game, $p_{m+1} \sqsubseteq c_{m,j}$. Since p_{m+1} is a complete prime, there exists $(e_1, \dots, e_n)$ that is j-reachable from (p_m, X_j) following $\mathcal{S}$ and such that $p_{m+1} \sqsubseteq e_j$. By definition of j-reachability, there exists a sequence of nodes $\widetilde{w}'_m$ such that $(p_m, X_j) \cdot \widetilde{w}'_m \cdot (p_{m+1}, X_j)$ conforms with $\mathcal{S}$. Furthermore $\widetilde{w}'_m$ consists of $V_O \cup \{\, (q, X_\ell) \mid j < \ell \,\}$ if $k \ge 1$. It is easy to see that (p_1, X_j) is reachable from $v'_0 = (d_1, \dots, d_n)$ following $\mathcal{S}$. Hence we have a sequence

$$v'_0 \cdot \widetilde{w}'_0 \cdot (p_1, X_j) \cdot \widetilde{w}'_1 \cdot (p_2, X_j) \cdot \widetilde{w}'_2 \cdot \dots$$

that conforms with $\mathcal{S}$ and starts from $v'_0 = (d_1, \dots, d_n)$. Since $\mathcal{S}$ is winning on v'_0, this sequence satisfies the parity condition, i.e. the priority of X_j is even. Hence the original play, which conforms with $\mathcal{S}'$, also satisfies the parity condition. □

We are now ready to prove soundness and completeness of the type-based characterization.

Theorem 7. *$p \sqsubseteq [\![\mathcal{E}]\!]_{\mathrm{L}}$ if and only if Proponent wins $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on (p, X_1).*

Proof. ($\Rightarrow$) By Lemma 20, Proponent wins $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on $(p, \bot, \dots, \bot)$. Since $(p, \bot, \dots, \bot) \cdot (p, X_1)$ is a valid play, Proponent also wins on (p, X_1).

($\Leftarrow$) If Proponent wins on (p, X_1), then Proponent also wins on (q, X_1) for every $q \sqsubseteq p$. Hence Proponent wins on $(p, \bot, \dots, \bot)$. By Lemma 23, we have a stable winning strategy $\mathcal{S}$ of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on $(d_1, \dots, d_n)$ with $p \sqsubseteq d_1$. Then $(d_1, \dots, d_n) \cdot (p, X_1)$ is a play that conforms with $\mathcal{S}$ and starts from $(d_1, \dots, d_n)$. Hence, by Lemma 22, $p \sqsubseteq g_1()$. By Lemma 19, we have $g_1() = [\![\mathcal{E}]\!]_{\mathrm{L}}$. Therefore, we have $p \sqsubseteq [\![\mathcal{E}]\!]_{\mathrm{L}}$ as required. □

D.4 The Opposite Parity Game

This subsection defines a game in which Proponent tries to disprove $p \sqsubseteq [\![\mathcal{E}]\!]_{\mathrm{L}}$ by giving an upper-bound $[\![\mathcal{E}]\!]_{\mathrm{L}} \sqsubseteq q$. A similar construction can be found in Salvati and Walukiewicz [48]; our construction can be seen as an infinite variant of them.

We first define the notion of complete coprimes, which is the dual of complete primes.

Definition 17 (Complete coprime). *Let $(A, \leq)$ be a complete lattice. An element $p \in A$ is a* complete coprime *if (1) $p \neq \top$ and (2) for every $U \subseteq A$, if $(\bigwedge_{x \in U} x) \leq p$, then $x \leq p$ for some $x \in U$.*

Lemma 24. *If $(A, \leq)$ is complete-prime algebraic, then for every $x \in A$,*

$$x = \bigwedge \{\, p \mid x \leq p,\ p\text{: complete coprime}\,\}.$$

Proof. Obviously $x \leq \bigwedge\{\, p \mid x \leq p,\ p\text{: complete coprime}\,\}$. We show that $x \geq \bigwedge\{\, p \mid x \leq p,\ p\text{: complete coprime}\,\}$. Since $(A, \leq)$ is complete-prime algebraic, by Lemma 16, it suffices to show that, for every complete prime q such that $q \not\leq x$, we have $q \not\leq \bigwedge\{\, p \mid x \leq p,\ p\text{: complete coprime}\,\}$, i.e., there exists a complete coprime $p \geq x$ such that $q \not\leq p$. Let q be a complete prime such that $q \not\leq x$. We show that

$$p = \bigvee\{\, q' \mid q \not\leq q',\ q'\text{: complete prime}\,\}$$

is indeed a complete coprime $p \geq x$ such that $q \not\leq p$. We first check $q \not\leq p$. If it were the case that $q \leq p$, then $q \leq p = \bigvee\{\, q' \mid q \not\leq q',\ q'\text{: complete prime}\,\}$, which would imply $q \leq q'$ for some q' with $q \not\leq q'$ (as q is a complete prime), a contradiction. It remains to check that p is a complete coprime. Since $q \not\leq p$, we have $p \neq \top$. Suppose that $\bigwedge_{y \in U} y \leq p$. If $q \leq y$ for every $y \in U$, then $q \leq p$, a contradiction. Hence $q \not\leq y$ for some $y \in U$. Now

$$\begin{aligned} y &= \bigvee\{\, q'' \mid q'' \leq y,\ q''\text{: complete prime}\,\} \\ &\leq \bigvee\{\, q'' \mid q \not\leq q'',\ q''\text{: complete prime}\,\} \\ &= p. \end{aligned}$$

Here the second inequality comes from the fact that $q'' \leq y$ implies $q \not\leq q''$; in fact, if $q'' \leq y$ and $q \leq q''$, then $q \leq q'' \leq y$, hence a contradiction. □

Definition 18 (Opposite parity game). *The parity game $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ is defined by the following data:*

$$\begin{aligned} V_P &:= \{\, (p, X_i) \mid p \in \mathcal{D}_{\mathrm{L},\tau_i},\ p\text{: complete coprime}\,\} \\ V_O &:= \{\, (d_1, \dots, d_n) \mid d_1 \in \mathcal{D}_{\mathrm{L},\tau_1}, \dots, d_n \in \mathcal{D}_{\mathrm{L},\tau_n}\,\} \\ E &:= \{\, ((p, X_i), (d_1, \dots, d_n)) \mid p \sqsupseteq [\![\varphi_i]\!]([X_1 \mapsto d_1, \dots, X_n \mapsto d_n])\,\} \\ &\quad \cup \{\, ((d_1, \dots, d_n), (p, X_i)) \mid p \sqsupseteq d_i,\ p\text{: complete coprime}\,\}. \end{aligned}$$

The priority of the opponent node is 0; the priority of node (p, X_i) is $2(n-i)+1$ if $\alpha_i = \nu$ and $2(n-i)+2$ if $\alpha_i = \mu$.

The game $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ is obtained by replacing complete primes with complete coprimes and $\sqsubseteq$ with $\sqsupseteq$. The position (p, X_i) is the state where Proponent ties to show that $[\![(\mathcal{E}, X_i)]\!] \sqsubseteq p$; to this end, Proponent picks a valuation $[X_1 \mapsto d_1, \dots, X_n \mapsto d_n]$ such that $(p \sqsupseteq [\![\varphi_i]\!]([X_1 \mapsto d_1, \dots, X_n \mapsto d_n])$. Opponent challenges the validity of the valuation $[X_1 \mapsto d_1, \dots, X_n \mapsto d_n]$, by picking i and a complete coprime p' such that $p' \sqsupseteq d_i$, and asking why $[\![(\mathcal{E}, X_i)]\!] \sqsubseteq p'$ holds.

Lemma 25. *If Proponent wins $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ on (p, X_i) (where p is a complete coprime), then for every complete prime $p' \in \mathcal{D}_{\mathrm{L},\tau_i}$ such that $p' \not\sqsubseteq p$, Opponent wins $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on (p', X_i).*

Proof. Let $\mathcal{S}$ be a P-winning strategy of $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ on (p, X_i) (where p is a complete coprime). We can assume without loss of generality that $\mathcal{S}$ is memoryless. Let $p' \in \mathcal{D}_{\mathrm{L},\tau_i}$ be a complete prime such that $p' \not\sqsubseteq p$.

Let (q, X_j) be a Proponent node in $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ and $(q', X_{j'})$ be a Proponent node in $\mathcal{G}_{\mathrm{L},\mathcal{E}}$. We say that (q, X_j) is *inconsistent with* $(q', X_{j'})$ if $j = j'$ and $q' \not\sqsubseteq q$. Similarly, given Opponent nodes $(d_1, \dots, d_n)$ in $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ and $(d'_1, \dots, d'_n)$ in $\mathcal{G}_{\mathrm{L},\mathcal{E}}$, those nodes are *inconsistent* if $d'_j \not\sqsubseteq d_j$ for some $1 \le j \le n$.

Let $(p', X_i) \cdot v'_1 \cdot v'_2 \cdot \dots \cdot v'_k$ be a play of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$. This play is *admissible* (with respect to $\mathcal{S}$) if there exists a play $(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_k$ of $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ that conforms with $\mathcal{S}$ such that v_j is inconsistent with v'_j for every $j \in \{1, \dots, k\}$. For each admissible play $(p', X_i) \cdot v'_1 \cdot v'_2 \cdot \dots \cdot v'_k$ of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$, we choose such a play $(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_k$ of $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$. Suppose that our choice satisfies the following property:

> If $(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_k$ is the chosen play corresponding to $(p', X_i) \cdot v'_1 \cdot v'_2 \cdot \dots \cdot v'_k$, then $(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_{k-1}$ is the chosen play for $(p', X_i) \cdot v'_1 \cdot v'_2 \cdot \dots \cdot v'_{k-1}$.

(A way to achieve this is to introduce a well-ordering on nodes of $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ and to choose the minimum play w.r.t. the lexicographic ordering from those that satisfy the requirement.)

We define an Opponent strategy $\overline{\mathcal{S}}$ for $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ defined on admissible plays starting from (p', X_i). Let

$$(p', X_i) \cdot v'_1 \cdot v'_2 \cdot \dots \cdot v'_k \cdot (d'_1, \dots, d'_n)$$

be an admissible play of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$. Let

$$(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_k \cdot (d_1, \dots, d_n)$$

be the chosen play of $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ corresponding to the above play. So v_j and v'_j are inconsistent for every $j \in \{1, \dots, k\}$ and $(d_1, \dots, d_n)$ is inconsistent with $(d'_1, \dots, d'_n)$. By definition, $d'_\ell \not\sqsubseteq d_\ell$ for some $1 \le \ell \le n$. Since $\mathcal{D}_{\mathrm{L},\tau_\ell}$ is complete-prime algebraic, by Lemma 24,

$$d'_\ell = \bigsqcup \{\, q' \in \mathcal{D}_{\mathrm{L},\tau_\ell} \mid q' \sqsubseteq d'_\ell,\ q' \text{: complete prime} \,\}$$

and

$$d_\ell = \bigsqcap \{\, q \in \mathcal{D}_{\mathrm{L},\tau_\ell} \mid d_\ell \sqsubseteq q,\ q \text{: complete coprime} \,\}.$$

Hence $d'_\ell \not\sqsubseteq d_\ell$ implies that there exist a complete prime $q' \sqsubseteq d'_\ell$ and a complete coprime $d_\ell \sqsubseteq q$ such that $q' \not\sqsubseteq q$. The Opponent strategy $\overline{\mathcal{S}}$ chooses (q', X_ℓ) as the next node in $\mathcal{G}_{\mathrm{L},\mathcal{E}}$.

We show that $\overline{\mathcal{S}}$ is indeed an O-strategy of $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ on (p', X_i). It suffices to show that every play $(p', X_i) \cdot v'_1 \cdot \dots \cdot v'_k$ that conforms with the Opponent strategy $\overline{\mathcal{S}}$ is admissible. If $k = 0$, the play is obviously admissible. Let $(p', X_i) \cdot v'_1 \cdot \dots \cdot v'_k$ be a play that conforms with $\overline{\mathcal{S}}$ and assume that it is admissible. If v'_k is an Opponent node, the next node v'_{k+1} is determined by $\overline{\mathcal{S}}$ and then $(p', X_i) \cdot v'_1 \cdot \dots \cdot v'_k \cdot v'_{k+1}$ is admissible by the definition of $\overline{\mathcal{S}}$. Consider the case that $v'_k = (p'_k, X_{i_k})$ is an Proponent node. Since this play is admissible, we have the chosen play $(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_k$ of $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ corresponding to the above play. Then v_j and v'_j are inconsistent for every $j = 1, \dots, k$. Since v_k is inconsistent with v'_k, $v_k = (p_k, X_{i_k})$ with $p'_k \not\sqsubseteq p_k$. Let $v'_{k+1} = (d'_1, \dots, d'_n)$ be an arbitrary Opponent node in $\mathcal{G}_{\mathrm{L},\mathcal{E}}$ such that (v_k, v_{k+1}) is valid in $\mathcal{G}_{\mathrm{L},\mathcal{E}}$. Let $v_{k+1} = (d_1, \dots, d_n)$ be the Opponent node in $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$ determined by $\mathcal{S}$. By definition of edges, $p'_k \sqsubseteq [\![\varphi_i]\!]([X_1 \mapsto d'_1, \dots, X_n \mapsto d'_n])$ and $[\![\varphi_i]\!]([X_1 \mapsto d_1, \dots, X_n \mapsto d_n]) \sqsubseteq p_k$. Since $[\![\varphi_i]\!]$ is monotone, $d'_j \sqsubseteq d_j$ for every $j = 1, \dots, n$ would imply $p'_k \sqsubseteq p_k$, a contradiction. Hence $d'_j \not\sqsubseteq d_j$ for some $1 \le j \le n$, that means, $(d'_1, \dots, d'_n)$ is inconsistent with $(d_1, \dots, d_n)$. Therefore the play $(p, X_i) \cdot v_1 \cdot v_2 \cdot \dots \cdot v_k \cdot (d'_1, \dots, d'_n)$ is admissible for every choice of $(d'_1, \dots, d'_n)$. So $\overline{\mathcal{S}}$ is defined for every play that conforms with $\overline{\mathcal{S}}$ and starts from (p', X_i).

We show that $\overline{\mathcal{S}}$ is O-winning. As $\overline{\mathcal{S}}$ does not get stuck, it suffices to show that every infinite play following $\overline{\mathcal{S}}$ does not satisfy the parity condition. Assume an infinite play

$$(p', X_i) \cdot v'_1 \cdot (p'_1, X_{i'_1}) \cdot v'_2 \cdot (p'_2, X_{i'_2}) \cdot \dots \cdot v'_\ell \cdot (p'_\ell, X_{i'_\ell}) \cdot \dots$$

following $\overline{\mathcal{S}}$. By the definition of $\overline{\mathcal{S}}$, every finite prefix of the play is admissible. Let $(p', X_i) \cdot v_{k,1} \cdot \dots \cdot v_{k,k}$ be the chosen play for the prefix of the above play of length k. Then, by the assumption on the choice, the choice of each node does not depend on the length of the prefix, i.e. $v_{k,i} = v_{k',i}$ for every $i \ge 1$ and $k, k' \ge i$. Hence the above infinite play is "admissible" in the sense that there exists an infinite play in $\overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}$

$$(p, X_i) \cdot v_1 \cdot (p_1, X_{i_1}) \cdot v_2 \cdot (p_2, X_{i_2}) \cdot \dots \cdot v_\ell \cdot (p_\ell, X_{i_\ell}) \cdot \dots$$

that conforms with $\mathcal{S}$ and such that (p_ℓ, X_{i_ℓ}) and $(p'_\ell, X_{i'_\ell})$ are inconsistent for every ℓ. Hence $i_\ell = i'_\ell$ for every ℓ. Because

$$(\text{priority of } (p_\ell, X_{i_\ell}) \text{ in } \overline{\mathcal{G}}_{\mathrm{L},\mathcal{E}}) = (\text{priority of } (p'_\ell, X_{i_\ell}) \text{ in } \mathcal{G}_{\mathrm{L},\mathcal{E}}) + 1,$$

the latter play satisfies the parity condition if and only if the former play does not satisfy the parity condition. Since $\mathcal{S}$ is a winning strategy, we conclude that the former play does not satisfy the parity condition. □

E Proof for Section 7

E.1 Some definitions

Here we introduce some notions and notations, which are useful in proofs of results of Section 7.

Definition 19 (Choice sequences). *Let π be an infinite sequence consisting of* L *and* R*, called a* choice sequence*. We define a reduction relation for pairs of terms and choice sequences by*

$$(t_1 \Box t_2;\, \mathtt{L}\pi) \xrightarrow{\epsilon}_D (t_1;\pi) \qquad (t_1 \Box t_2;\, \mathtt{R}\pi) \xrightarrow{\epsilon}_D (t_2;\pi)$$

and $(u;\pi) \xrightarrow{\ell}_D (u';\pi)$ if $u \neq u_1 \Box u_2$ and $u \xrightarrow{\ell}_D u'$. We sometimes omit labels and just write $(u;\pi) \longrightarrow_D (u';\pi)$ for $(u;\pi) \xrightarrow{\ell}_D (u';\pi)$. A choice sequence resolves nondeterminism of a program P; a reduction sequence is completely determined by its initial term, its length and a choice sequence. We write $\longrightarrow_D^k$ for the k-step reduction relation. If $(t_0,\pi_0) \longrightarrow_D (t_1,\pi_1) \longrightarrow_D \cdots \longrightarrow_D (t_k,\pi_k)$, then we write $\pi_0 \Vdash t_0 \longrightarrow_D t_1 \longrightarrow_D \cdots \longrightarrow_D t_k$ and say that the reduction sequence $t_0 \longrightarrow_D t_1 \longrightarrow_D \cdots \longrightarrow_D t_k$ follows π_0.

We introduce a slightly more elaborate notion of the recursive call relation below. Note that the relation $f\,\widetilde{t} \rightsquigarrow_D g\,\widetilde{u}$ in Definition 6 coincides with $f\,\widetilde{t} \rightsquigarrow_D g\,\widetilde{u}$ redefined below.

Definition 20 (Recursive call relation, call sequences). *Let $P = (D, \mathbf{main})$ be a program, with $D = \{f_i\,\widetilde{x_i} = u_i\}_{1\le i\le n}$. We define $D^\sharp := D \cup \{f_i^\sharp\,\tilde{x} = u_i\}_{1\le i\le n}$ where $f_1^\sharp,\ldots,f_n^\sharp$ are fresh symbols. So $D^\sharp$ has two copies of each function symbol, one of which is marked. For the terms $\widetilde{t_i}$ and $\widetilde{t_j}$ that do not contain marked symbols, we write $(f_i\,\widetilde{t_i};\pi) \overset{k+1}{\rightsquigarrow}_D (f_j\,\widetilde{t_j};\pi')$ if*

$$([\widetilde{t_i}/\widetilde{x_i}][f_1^\sharp/f_1,\ldots,f_n^\sharp/f_n]u_i;\ \pi) \xrightarrow{k}_{D^\sharp} (f_j^\sharp\,\widetilde{t'_j};\ \pi')$$

(then $\widetilde{t_j}$ is obtained by erasing all the marks in $\widetilde{t'_j}$). If there exists a (finite or infinite) sequence $(f\,\tilde{s};\pi) \overset{k_1}{\rightsquigarrow}_D (g_1\,\tilde{u}_1;\pi_1) \overset{k_2}{\rightsquigarrow}_D (g_2\,\tilde{u}_2;\pi_2) \overset{k_3}{\rightsquigarrow}_D \cdots$, then we write $\pi \Vdash f\,\tilde{s} \overset{k_1}{\rightsquigarrow}_D g_1\,\tilde{u}_1 \overset{k_2}{\rightsquigarrow}_D g_2\,\tilde{u}_2 \overset{k_3}{\rightsquigarrow}_D \cdots$ and call it a call sequence of $f\,\tilde{s}$ following π*. We often omit the number k_j of steps and the choice sequence π. Given a program $P = (D, \mathbf{main})$, the* set of call sequences of P *is a subset of (finite or infinite) sequences of function symbols defined by $\mathbf{Callseq}(P) := \{\,\mathbf{main}\,g_1\,g_2\cdots \mid \mathbf{main} \rightsquigarrow_D g_1\,\widetilde{u_1} \rightsquigarrow_D g_2\,\widetilde{u_2} \rightsquigarrow_D \cdots\}$.*

Note that by the definition above, $(f_i\,\widetilde{t_i};\pi) \overset{k}{\rightsquigarrow}_D (f_j\,\widetilde{t_j};\pi')$ implies $(f_i\,\widetilde{t_i};\pi) \longrightarrow_D^k (f_j\,\widetilde{t_j};\pi')$. We write $t^\sharp$ for the term obtained by marking all function symbols in t, i.e. $[f_1^\sharp/f_1,\ldots,f_n^\sharp/f_n]t$.

E.2 Existence of a unique infinite call-sequence

Let $P = (D, \mathbf{main})$ be a program and

$$\pi \Vdash \mathbf{main} \xrightarrow{\ell_1}_D t_1 \xrightarrow{\ell_2}_D \cdots$$

be the infinite reduction sequence following π. This subsection proves that there exists a unique infinite call-sequence

$$\pi \Vdash f\,\tilde{u} \rightsquigarrow_D g_1\,\tilde{u}_1 \rightsquigarrow_D g_2\,\tilde{u}_2 \rightsquigarrow_D \cdots$$

of this reduction sequence.

Lemma 26. *Let $P = (D, \mathbf{main})$ be a program and $\pi \in \{\mathtt{L}, \mathtt{R}\}^\omega$ be a choice sequence. Suppose we have an infinite reduction sequence*

$$\pi \Vdash \mathbf{main} \longrightarrow_D t_1 \longrightarrow_D t_2 \longrightarrow_D \cdots.$$

Then there exists an infinite call-sequence

$$\pi \Vdash \mathbf{main} \rightsquigarrow_D g_1\,\tilde{u}_1 \rightsquigarrow_D g_2\,\tilde{u}_2 \rightsquigarrow_D \cdots.$$

Proof. Kobayashi and Ong [47, Appendix B] have proved a similar result for simply-typed programs without integers. We remove integers from D as follows:

– First, replace each conditional branching **if** $p(t'_1, \ldots, t'_k)$ **then** t_1 **else** t_2 with the nondeterministic branching $t_1 \Box t_2$.
– Then replace each integer expression with the unit value.

Let us write D' for the resulting function definitions. Then there exists a choice π' such that $\pi' \Vdash \mathbf{main} \longrightarrow_{D'} t'_1 \longrightarrow_{D'} t'_2 \longrightarrow_{D'} \cdots$ and, for every $i \geq 1$, t'_i is obtained from t_i by the above translation. By the result of Kobayashi and Ong, this sequence has an infinite call-sequence. This call-sequence can be transformed into a call-sequence of the original reduction sequence. $\square$

We prove uniqueness.

Suppose that $(f\,\widetilde{u}; \pi) \longrightarrow_D^k (g\,\widetilde{v}; \pi')$. Given an occurrence of g in $f\,\widetilde{u}$, we write $f'\,\widetilde{u}'$ for the term obtained by replacing the occurrence of g with $g^\sharp$. We say that the head-occurrence g in $g\,\widetilde{v}$ is a *copy of the occurrence of g in $f\,\widetilde{u}$* if $(f'\,\widetilde{u}'; \pi) \longrightarrow_{D^\sharp}^k (g^\sharp\,\widetilde{v}'; \pi)$. An occurrence of h in $f\,\widetilde{u}$ is an *ancestor of the head-occurrence of g in $g\,\widetilde{v}$* if

$$(f\,\widetilde{u}; \pi) \longrightarrow_D^{k_0} (h_1\,\widetilde{w}_1; \pi_1) \overset{k_1}{\rightsquigarrow}_D (h_2\,\widetilde{w}_2; \pi_2) \overset{k_2}{\rightsquigarrow}_D \ldots \overset{k_\ell}{\rightsquigarrow}_D (h_\ell\,\widetilde{w}_\ell; \pi_\ell) = (g\,\widetilde{v}; \pi')$$

where $\ell \geq 1$, $k = \sum_{j=0}^{\ell} k_j$ and h_1 is a copy of the occurrence of h in $f\,\widetilde{u}$ (hence $h_1 = h$ as function symbols). It is not difficult to see that, given a reduction sequence $(f\,\widetilde{u}; \pi) \longrightarrow_D^k (g\,\widetilde{v}; \pi')$, there exists a unique ancestor of g in $f\,\widetilde{u}$.

Lemma 27. *For every term t and choice sequence $\pi \in \{\mathtt{L}, \mathtt{R}\}^\omega$, there exists at most one infinite call-sequence of t following π.*

Proof. Let $\pi \Vdash t_0 \longrightarrow_D t_1 \longrightarrow_D t_2 \longrightarrow_D \cdots$ be the infinite reduction sequence following π. In this proof, we shall consider only sequences following π and hence we shall omit $\pi \Vdash$.

Assume that there exist difference infinite call-sequences, say,

$$t_0 \rightsquigarrow_D g_1\,\tilde{u}_1 \rightsquigarrow_D \cdots \rightsquigarrow_D g_k\,\tilde{u}_k \rightsquigarrow_D h_1\,\tilde{v}_1 \rightsquigarrow_D h_2\,\tilde{v}_2 \rightsquigarrow_D \cdots$$

and

$$t_0 \rightsquigarrow_D g_1\,\tilde{u}_1 \rightsquigarrow_D \cdots \rightsquigarrow_D g_k\,\tilde{u}_k \rightsquigarrow_D k_1\,\tilde{w}_1 \rightsquigarrow_D k_2\,\tilde{w}_2 \rightsquigarrow_D \cdots$$

where h_1 and k_1 are different. Since both call sequences are infinite, the reduction sequence has infinite switching of the sequence, i.e. the reduction sequence must be of the following form:

$$t_0 \longrightarrow_D^* h_{i_1}\,\tilde{v}_{i_1} \longrightarrow_D^* k_{j_1}\,\tilde{w}_{j_1} \longrightarrow_D^* h_{i_2}\,\tilde{v}_{i_2} \longrightarrow_D^* k_{j_2}\,\tilde{w}_{j_2} \longrightarrow_D^* \cdots.$$

Since h_{i_1} is not an ancestor of k_{j_1}, an ancestor of k_{j_1} must appear in $\tilde{v}_{i_1}$, say, in the ℓ_1-th argument v_{i_1,ℓ_1}. Since k_{j_1} appears in the reduction sequence, v_{i_1,ℓ_1} must have a head occurrence and the reduction sequence is of the form

$$t_0 \longrightarrow_D^* h_{i_1}\,\tilde{v}_{i_1} \longrightarrow_D^* v_{i_1,\ell_1}\,\tilde{s}_1 \longrightarrow_D^* k_{j_1}\,\tilde{w}_{j_1} \longrightarrow_D^* h_{i_2}\,\tilde{v}_{i_2} \longrightarrow_D^* \cdots.$$

Since h_{i_1} is an ancestor of h_{i_2}, there is no ancestor of h_{i_2} in $\tilde{v}_{i_1}$, in particular, in v_{i_1,ℓ_1}. So an ancestor of h_{i_2} must appear in $\tilde{s}_1$, say, s_{1,ℓ_2}, and the reduction sequence is of the form

$$t_0 \longrightarrow_D^* h_{i_1}\,\tilde{v}_{i_1} \longrightarrow_D^* v_{i_1,\ell_1}\,\tilde{s}_1 \longrightarrow_D^* s_{1,\ell_2}\,\tilde{s}_2 \longrightarrow_D^* h_{i_2}\,\tilde{v}_{i_2} \longrightarrow_D^* k_{j_2}\,\tilde{w}_{j_2} \longrightarrow_D^* \cdots.$$

Since an ancestor of k_{j_2} appears in v_{i_1,ℓ_1}, there is no ancestor in $\tilde{s}_1$, in particular, in s_{1,ℓ_2}. Hence there is an ancestor of k_{j_2} in $\tilde{s}_2$, say, s_{2,ℓ_3}, and the reduction sequence is

$$t_0 \longrightarrow_D^* h_{i_1}\,\tilde{v}_{i_1} \longrightarrow_D^* v_{i_1,\ell_1}\,\tilde{s}_1 \longrightarrow_D^* s_{1,\ell_2}\,\tilde{s}_2 \longrightarrow_D^* s_{2,\ell_3}\,\tilde{s}_3 \longrightarrow_D^* k_{j_2}\,\tilde{w}_{j_2} \longrightarrow_D^* h_{i_3}\,\tilde{v}_{i_3} \longrightarrow_D^* \cdots.$$

By repeatedly applying the above argument, we have the following decomposition of the reduction sequence:

$$t_0 \longrightarrow_D^* h_{i_1}\,\tilde{v}_{i_1} \longrightarrow_D^* v_{i_1,\ell_1}\,\tilde{s}_1 \longrightarrow_D^* s_{1,\ell_2}\,\tilde{s}_2 \longrightarrow_D^* s_{2,\ell_3}\,\tilde{s}_3 \longrightarrow_D^* s_{3,\ell_4}\,\tilde{s}_4 \longrightarrow_D^* s_{4,\ell_5}\,\tilde{s}_5 \longrightarrow_D^* \cdots.$$

Then the order of s_{m,ℓ_m} is greater than $s_{m+1,\ell_{m+1}}$ for every m since $s_{m+1,\ell_{m+1}}$ is an argument of s_{m,ℓ_m}. This contradicts the assumption that the program D is simply typed. □

Corollary 1. *Let $P = (D, \mathbf{main})$ be a program and*

$$\pi \Vdash \mathbf{main} \longrightarrow_D t_1 \longrightarrow_D \cdots$$

be the infinite reduction sequence following π. Then there exists a unique infinite call-sequence of this reduction sequence, i.e.

$$\pi \Vdash \mathbf{main} \rightsquigarrow_D g_1\,\tilde{u}_1 \rightsquigarrow_D g_2\,\tilde{u}_2 \rightsquigarrow_D \cdots.$$

E.3 Proof of Theorem 4

Let $\mathtt{L}_0 = (\{\mathtt{s}_\star\}, \emptyset, \emptyset, \mathtt{s}_\star)$ be the transition system with one state and no transition. This section shows that given a program $P = (D, \mathbf{main})$ with priority Ω, $\models_{csa} (P, \Omega)$ holds if and only if $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$. Without loss of generality, we assume below that P is of the form $(D, \mathbf{main})$; note that, if $P = (D, t)$, then we can replace it with $(D \cup \{\mathbf{main} = t\}, \mathbf{main})$.

Fix a program $P = (D, \mathbf{main})$ with priority Ω. Suppose that $D = \{f_1\,\tilde{x}_1 = t_1, \ldots, f_n\,\tilde{x}_n = t_n\}$ and let κ_i be the simple type of f_i. We can assume without loss of generality that $\mathbf{main} = f_1$ and that $\Omega(f_i) \geq \Omega(f_j)$ if $i \geq j$. Let

$$\Omega'(f_i) = \begin{cases} 2(n-i) & (\text{if } \Omega(f_i) \text{ is even}) \\ 2(n-i)+1 & (\text{if } \Omega(f_i) \text{ is odd}). \end{cases}$$

The priority assignments Ω and Ω' are equivalent in the sense that an infinite sequence $g_1 g_2 \ldots$ of function symbols satisfies the parity condition with respect to Ω if and only if so does to Ω'. Hence we can assume without loss of generality that $2(n-i) \leq \Omega(f_i) \leq 2(n-i)+1$.

Given a type κ (resp. η) of the target language, we define a type $\kappa^{\dagger_{csa}}$ (resp. $\eta^{\dagger_{csa}}$) for HFLs as follows:

$$\star^{\dagger_{csa}} := \bullet \qquad (\eta \to \kappa)^{\dagger_{csa}} := \eta^{\dagger_{csa}} \to \kappa^{\dagger_{csa}} \qquad \mathtt{int}^{\dagger_{csa}} := \mathtt{int}.$$

Simply $(-)^{\dagger_{csa}}$ replaces $\star$ with $\bullet$. The operation $(-)^{\dagger_{csa}}$ is pointwise extended for type environments by: $(x_1 : \eta_1, \ldots, x_\ell : \eta_\ell)^{\dagger_{csa}} = x_1 : \eta_1{}^{\dagger_{csa}}, \ldots, x_\ell : \eta_\ell{}^{\dagger_{csa}}$ Given an LTS $\mathtt{L}$, let us define $\mathcal{D}_{\mathtt{L},\kappa} := \mathcal{D}_{\mathtt{L},\kappa^{\dagger_{csa}}}$.

Definition 21 (Semantic interpretation of a program). *Let $\mathcal{K} \vdash t : \kappa$. A* valuation *of $\mathcal{K}$ is a mapping ρ such that $dom(\rho) = dom(\mathcal{K})$ and $\rho(x) \in \mathcal{D}_{\mathtt{L},\mathcal{K}(x)}$ for each $x \in dom(\mathcal{K})$. The set of valuations of $\mathcal{K}$ is ordered by the point-wise ordering. The* interpretation *of a type judgment $\mathcal{K} \vdash t : \eta$ is a (monotonic) function from the valuations of $\mathcal{K}$ to $\mathcal{D}_{\mathtt{L},\kappa}$ inductively defined by:*

$$\begin{aligned}
[\![\mathcal{K} \vdash () : \star]\!](\rho) &:= \{\mathtt{s}_\star\} \\
[\![\mathcal{K} \vdash x : \kappa]\!](\rho) &:= \rho(x) \\
[\![\mathcal{K} \vdash n : \mathtt{int}]\!](\rho) &:= n \\
[\![\mathcal{K} \vdash t_1 \,\mathtt{op}\, t_2 : \mathtt{int}]\!](\rho) &:= ([\![\mathcal{K} \vdash t_1 : \mathtt{int}]\!](\rho))[\![\mathtt{op}]\!]([\![\mathcal{K} \vdash t_2 : \mathtt{int}]\!](\rho)) \\
[\![\mathcal{K} \vdash \mathbf{if}\ p(t'_1, \ldots, t'_k)\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 : \star]\!](\rho) &:= \begin{cases} [\![\mathcal{K} \vdash t_1 : \star]\!](\rho) & (\textit{if } ([\![t'_1]\!](\rho), \ldots, [\![t'_k]\!](\rho)) \in [\![p]\!]) \\ [\![\mathcal{K} \vdash t_2 : \star]\!](\rho) & (\textit{if } ([\![t'_1]\!](\rho), \ldots, [\![t'_k]\!](\rho)) \notin [\![p]\!]) \end{cases} \\
[\![\mathcal{K} \vdash \mathbf{event}\ a; t : \star]\!](\rho) &:= [\![\mathcal{K} \vdash t : \star]\!](\rho) \\
[\![\mathcal{K} \vdash t_1\, t_2 : \kappa]\!](\rho) &:= ([\![\mathcal{K} \vdash t_1 : \eta \to \kappa]\!](\rho))([\![\mathcal{K} \vdash t_2 : \eta]\!](\rho)) \\
[\![\mathcal{K} \vdash t_1 \Box t_2 : \star]\!](\rho) &:= [\![\mathcal{K} \vdash t_1 : \star]\!](\rho) \sqcap [\![\mathcal{K} \vdash t_2 : \star]\!](\rho) \\
[\![\mathcal{K} \vdash \lambda x.t : \eta \to \kappa]\!](\rho) &:= \{(d, [\![\mathcal{K}, x : \eta \vdash t : \kappa]\!](\rho[x \mapsto d])) \mid d \in \mathcal{D}_{\mathtt{L},\eta}\}.
\end{aligned}$$

We often write just $[\![t]\!]$ for $[\![\mathcal{K} \vdash t : \eta]\!]$ (on the assumption that the simple type environment for t is implicitly determined).

Lemma 28. $[\![\mathcal{K} \vdash t : \eta]\!]_{\mathtt{L}_0}([x_1 \mapsto d_1, \ldots, x_n \mapsto d_n]) = [\![\mathcal{K}^{\dagger_{csa}} \vdash t^{\dagger_{csa}} : \eta^{\dagger_{csa}}]\!]_{\mathtt{L}_0}([x_1 \mapsto d_1, \ldots, x_n \mapsto d_n])$.

Proof. By induction on the structure of t. □

Completeness We show that $\models_{csa} (P, \Omega)$ implies $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$. We use game-based characterization of $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$.

Definition 22 (Parity game for a program). *The parity game $\mathcal{G}_{\mathtt{L},(P,\Omega)}$ is defined as follows:*

$$\begin{aligned} V_P &:= \{\, (p, f_i) \mid p \in \mathcal{D}_{\mathtt{L},\kappa_i},\ p\text{: complete prime} \,\} \\ V_O &:= \{\, (d_1, \ldots, d_n) \mid d_1 \in \mathcal{D}_{\mathtt{L},\kappa_1}, \ldots, d_n \in \mathcal{D}_{\mathtt{L},\kappa_n} \,\} \\ E &:= \{\, ((p, f_i), (d_1, \ldots, d_n)) \mid p \sqsubseteq [\![\lambda \tilde{x}.t_i]\!]([f_1 \mapsto d_1, \ldots, f_n \mapsto d_n]) \,\} \\ &\quad \cup \{\, ((d_1, \ldots, d_n), (p, f_i)) \mid p \sqsubseteq d_i,\ p\text{: complete prime} \,\}. \end{aligned}$$

The priority of $(d_1, \ldots, d_n)$ is 0 and that of (p, f_i) is the priority $\Omega(f_i)$ of f_i.

Corollary 2. *$\mathcal{G}_{\mathtt{L}_0,(P,\Omega)}$ is isomorphic to $\mathcal{G}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$, i.e. there exists a bijection of nodes that preserves the owner and the priority of each node.*

Proof. The bijection on nodes is given by $(d_1, \ldots, d_n) \leftrightarrow (d_1, \ldots, d_n)$ and $(p, f_i) \leftrightarrow (p, f_i)$. This bijection preserves edges because of Lemma 28. □

We define a strategy of $\mathcal{G}_{\mathtt{L}_0,(P,\Omega)}$ on $(\{\mathtt{s}_\star\}, f_1)$ by inspecting the reduction sequences of P. The technique used here is the same as [22]. In this construction, we need to track occurrences of a function symbol or a term in a reduction sequence. This can be achieved by marking symbols and terms if needed as in Definition 6. In the sequel, we shall not introduce explicit marking and use the convention that, if the same metavariable appears in a reduction sequence (e.g. $f\,u \longrightarrow^* u\,\widetilde{v}$), then the first one is the origin of all others.

The occurrence of t in $t\,u_1\,\ldots\,u_k$ $(k \geq 0)$ is called a *head occurrence*. Given a head occurrence of t of a term reachable from **main** (i.e. **main** $\longrightarrow_D^* t\,\tilde{u}$ for some $\tilde{u}$), we assign an element $d_{t,\tilde{u}} \in \mathcal{D}_{\mathtt{L}_0,\kappa}$ where κ is the simple type of t by induction on the order of κ.

- Case $\widetilde{u}$ is empty: Then $t : \star$ and we define $d_{t,\epsilon} = \{\mathtt{s}_\star\}$.
- Case $t\,\tilde{u} = t\,u_1\,\ldots\,u_k$: Note that the order of u_j $(1 \leq j \leq k)$ is less than that of t. For every $1 \leq j \leq k$, we define e_j as follows.
  - Case $u_j : \mathtt{int}$: Then $e_j = [\![u_j]\!]$. (Note that u_j has no free variable and no function symbol.)
  - Case $u_j : \kappa$: Let e_j be the element defined by

$$e_j = \bigsqcup \{\, d_{u_j,\tilde{v}} \mid t\,\tilde{u} \longrightarrow_D^* u_j\,\tilde{v} \,\}.$$

Then $d_{t,\tilde{u}}$ is the function defined by

$$d_{t,\tilde{u}}(x_1,\dots,x_k) := \begin{cases} \{\mathtt{s}_\star\} & (\text{if } e_j \sqsubseteq x_j \text{ for all } j \in \{1,\dots,k\}) \\ \{\,\} & (\text{otherwise}). \end{cases}$$

Note that $d_{t,\tilde{u}}$ is the minimum function such that $d_{t,\tilde{u}}(e_1,\dots,e_k) = \{\mathtt{s}_\star\}$.

Let $\theta = [s_1/x_1,\dots,s_k/x_k]$ and assume **main** $\longrightarrow_D^* (\theta t_0)\,\tilde{u}$. We define a mapping from function symbols and variables to elements of their semantics domain as follows.

- For a function symbol f_i:

$$\rho_{t_0,\tilde{u},\theta}(f_i) = \bigsqcup\{\, d_{f_i,\tilde{v}} \mid (\theta t_0)\widetilde{u} \longrightarrow_D^* f_i\tilde{v},\ f_i \text{ originates from } t_0 \,\}$$

- For a variable $x_i : \mathtt{int}$:

$$\rho_{t_0,\tilde{u},\theta}(x_i) = [\![s_i]\!]$$

- For a variable $x_i : \kappa$:

$$\rho_{t_0,\tilde{u},\theta}(x_i) = \bigsqcup\{\, d_{s_i,\tilde{v}} \mid (\theta t_0)\widetilde{u} \longrightarrow_D^* s_i\tilde{v},\ s_i \text{ originates from } \theta(x_i) \,\}.$$

Lemma 29. *Let $\theta = [s_1/x_1,\dots,s_k/x_k]$ and assume* **main** $\longrightarrow_D^* (\theta t_0)\,\tilde{u}$. *Let v be an arithmetic expression (i.e. a term of type* `int`*) appearing in t_0. Then $[\![\theta v]\!] = [\![v]\!](\rho_{t_0,\tilde{u},\theta})$.*

Proof. By induction on v.

- Case $v = x_i$: Then $[\![\theta x_i]\!] = [\![s_i]\!] = \rho_{t_0,\tilde{u},\theta}(x_i) = [\![x_i]\!](\rho_{t_0,\tilde{u},\theta})$.
- Case $v = v_1 \mathtt{op} v_2$: By the induction hypothesis, $[\![\theta v_j]\!] = [\![v_j]\!](\rho_{t_0,\tilde{u},\theta})$ for $j = 1,2$. Then $[\![\theta(v_1 \mathtt{ op } v_2)]\!] = [\![\theta v_1]\!][\![\mathtt{op}]\!][\![\theta v_2]\!] = [\![v_1]\!](\rho_{t_0,\tilde{u},\theta})[\![\mathtt{op}]\!][\![v_2]\!](\rho_{t_0,\tilde{u},\theta}) = [\![v_1 \mathtt{ op } v_2]\!](\rho_{t_0,\tilde{u},\theta})$.

□

Lemma 30. *Let $\theta = [s_1/x_1,\dots,s_k/x_k]$ and assume* **main** $\longrightarrow_D^* (\theta t_0)\,\tilde{u}$. *Then $d_{\theta t_0,\tilde{u}} \sqsubseteq [\![t_0]\!](\rho_{t_0,\tilde{u},\theta})$.*

Proof. By induction on the structure of t_0.

- Case $t_0 = (\,)$: Then $[\![t_0]\!] = \{\mathtt{s}_\star\} = d_{(),\epsilon}$.
- Case $t_0 = x_i$ for some $1 \le i \le k$: Then $(\theta t_0)\tilde{u} = s_i\,\tilde{u}$. By definition, $d_{s_i,\tilde{u}} = \rho_{t_0,\tilde{u},\theta}(x_i)$.
- Case $t_0 = f_i$: Then $(\theta t_0)\tilde{u} = f_i\,\tilde{u}$. Since this occurrence of f_i originates from t_0, we have $d_{f_i,\tilde{u}} = \rho_{t_0,\tilde{u},\theta}(f_i)$.
- Case $t_0 = n$ or $t_0 = t_1 \mathtt{ op } t_2$: Never occurs because the type of t_0 must be of the form $\eta_1 \to \cdots \to \eta_\ell \to \star$.

– Case $t_0 = \textbf{if}\ p(t'_1, \ldots, t'_k)\ \textbf{then}\ t_1\ \textbf{else}\ t_2$: Then $\tilde{u}$ is an empty sequence. Assume that $([\![\theta t'_1]\!], \ldots, [\![\theta t'_k]\!]) \in [\![p]\!]$. The other case can be proved in a similar manner.
  Then we have $\theta t_0 \longrightarrow_D \theta t_1$. By definition, $d_{\theta t_0,\epsilon} = d_{\theta t_1,\epsilon} = \{\mathtt{s}_\star\}$.
  Since every reduction sequence $\theta t_0 \longrightarrow^*_D s\,\tilde{v}$ can be factored into $\theta t_0 \longrightarrow_D \theta t_1 \longrightarrow^*_D s\,\tilde{v}$, we have $\rho_{t_0,\epsilon,\theta} = \rho_{t_1,\epsilon,\theta}$. By the induction hypothesis, $\{\mathtt{s}_\star\} \sqsubseteq [\![t_1]\!](\rho_{t_1,\epsilon,\theta})$. Hence $\{\mathtt{s}_\star\} \sqsubseteq [\![t_1]\!](\rho_{t_0,\epsilon,\theta})$.
  By Lemma 29, $[\![\theta t'_j]\!] = [\![t'_j]\!](\rho_{t_0,\tilde{u},\theta})$ for every $j = 1, \ldots, k$. Hence $([\![\theta t'_1]\!], \ldots, [\![\theta t'_k]\!]) \in [\![p]\!]$ implies $([\![t'_1]\!](\rho_{t_0,\tilde{u},\theta}), \ldots, [\![t'_k]\!](\rho_{t_0,\tilde{u},\theta})) \in [\![p]\!]$. By definition, $[\![t_0]\!](\rho_{t_0,\tilde{u},\theta}) = [\![t_1]\!](\rho_{t_0,\tilde{u},\theta}) \sqsupseteq \{\mathtt{s}_\star\}$.
– Case $t_0 = \textbf{event}\ a; t$: Then t_0 has type $\star$ and $\tilde{u}$ is the empty sequence. We have $\theta t_0 = \textbf{event}\ a; (\theta t)$. Since every reduction sequence $\theta t_0 \longrightarrow^*_D s\,\tilde{v}$ can be factored into $\theta t_0 \longrightarrow_D \theta t \longrightarrow^*_D s\,\tilde{v}$, we have $\rho_{t_0,\epsilon,\theta} = \rho_{t,\epsilon,\theta}$. By definition, $d_{\theta t_0,\epsilon} = d_{\theta t,\epsilon} = \{\mathtt{s}_\star\}$ and $[\![t_0]\!](\rho_{t_0,\epsilon,\theta}) = [\![t]\!](\rho_{t_0,\epsilon,\theta}) = [\![t]\!](\rho_{t,\epsilon,\theta})$. By the induction hypothesis, $d_{\theta t,\epsilon} \sqsubseteq [\![t]\!](\rho_{t,\epsilon,\theta})$. Hence $d_{\theta t_0,\epsilon} \sqsubseteq [\![t_0]\!](\rho_{t_0,\epsilon,\theta})$.
– Case $t_0 = t_1\,t_2$: Suppose that $(\theta t_1)(\theta t_2)\tilde{u} \longrightarrow^*_D (\theta t_2)\,\tilde{v}$. We first show that $\rho_{t_2,\tilde{v},\theta}(x) \sqsubseteq \rho_{t_1 t_2,\tilde{u},\theta}(x)$ for every $x \in \{f_1, \ldots, f_n\} \cup \{x_1, \ldots, x_k\}$.
  • Case $x = x_i : \mathtt{int}$: Then $\rho_{t_2,\tilde{v},\theta}(x_i) = \rho_{t_1 t_2,\tilde{u},\theta}(x_i) = [\![s_i]\!]$.
  • Case $x = x_i : \kappa$: Then

$$\begin{aligned}\rho_{t_2,\tilde{v},\theta}(x_i) &= \bigsqcup\{\, d_{s_i,\tilde{w}} \mid (\theta t_2)\,\tilde{v} \longrightarrow^*_D s_i\tilde{w},\ s_i \text{ originates from } \theta(x_i)\,\}\\ &\sqsubseteq \bigsqcup\{\, d_{s_i,\tilde{w}} \mid (\theta(t_1\,t_2))\,\tilde{u} \longrightarrow^*_D s_i\tilde{w},\ s_i \text{ originates from } \theta(x_i)\,\}\\ &= \rho_{t_1 t_2,\tilde{u},\theta}(x_i)\end{aligned}$$

  because $(\theta t_2)\,\tilde{v} \longrightarrow^*_D s_i\tilde{w}$ (where s_i originates from θx_i) implies $(\theta(t_1\,t_2))\,\tilde{u} \longrightarrow^*_D (\theta t_2)\tilde{v} \longrightarrow^*_D s_i\tilde{w}$ (where s_i originates from θx_i).
  • Case $x = f_i$:

$$\begin{aligned}\rho_{t_2,\tilde{v},\theta}(f_i) &= \bigsqcup\{\, d_{f_i,\tilde{w}} \mid (\theta t_2)\,\tilde{v} \longrightarrow^*_D f_i\tilde{w},\ f_i \text{ originates from } t_2\,\}\\ &\sqsubseteq \bigsqcup\{\, d_{f_i,\tilde{w}} \mid (\theta(t_1\,t_2))\,\tilde{u} \longrightarrow^*_D f_i\tilde{w},\ f_i \text{ originates from } t_1 t_2\,\}\\ &= \rho_{t_1 t_2,\tilde{u},\theta}(f_i)\end{aligned}$$

  because $(\theta t_2)\,\tilde{v} \longrightarrow^*_D f_i\tilde{w}$ (where f_i originates from t_2) implies $(\theta(t_1\,t_2))\,\tilde{u} \longrightarrow^*_D (\theta t_2)\tilde{v} \longrightarrow^*_D f_i\tilde{w}$ (where f_i originates from $t_1 t_2$).
  It is easy to see that $\rho_{t_1,(\theta t_2)\,\tilde{u},\theta} \sqsubseteq \rho_{t_1 t_2,\tilde{u},\theta}$. By the induction hypothesis,

$$d_{\theta t_1,(\theta t_2)\,\tilde{u}} \sqsubseteq [\![t_1]\!](\rho_{t_1,(\theta t_2)\,\tilde{u},\theta}) \sqsubseteq [\![t_1]\!](\rho_{t_1 t_2,\tilde{u},\theta}).$$

  By the definition of $d_{\theta t_1,(\theta t_2)\,\tilde{u}}$,

$$d_{\theta t_1,(\theta t_2)\,\tilde{u}}\Big(\bigsqcup\{d_{\theta t_2,\tilde{v}} \mid (\theta t_1)\,(\theta t_2)\,\tilde{v} \longrightarrow^*_D (\theta t_2)\tilde{v}\}\Big) = d_{(\theta t_1)\,(\theta t_2),\,\tilde{u}}.$$

  By the induction hypothesis, for every reduction sequence $(\theta t_1)\,(\theta t_2)\,\tilde{u} \longrightarrow^*_D (\theta t_2)\tilde{v}$, we have $d_{\theta t_2,\tilde{v}} \sqsubseteq [\![t_2]\!](\rho_{t_2,\tilde{v},\theta})$. Since $\rho_{t_2,\tilde{v},\theta}(x) \sqsubseteq \rho_{t_1 t_2,\tilde{u},\theta}(x)$ for every x, we have

$$d_{\theta t_2,\tilde{v}} \sqsubseteq [\![t_2]\!](\rho_{t_2,\tilde{v},\theta}) \sqsubseteq [\![t_2]\!](\rho_{t_1 t_2,\tilde{u},\theta}).$$

Because the reduction sequence $(\theta t_1)\,(\theta t_2)\,\tilde{u} \longrightarrow_D^* (\theta t_2)\tilde{v}$ is arbitrary,

$$\Big(\bigsqcup\{d_{\theta t_2,\tilde{v}} \mid (\theta t_1)\,(\theta t_2)\,\tilde{v} \longrightarrow_D^* (\theta t_2)\tilde{v}\}\Big) \sqsubseteq [\![t_2]\!](\rho_{t_1 t_2,\tilde{u},\theta}).$$

Therefore, by monotonicity,

$$\begin{aligned} d_{(\theta t_1)\,(\theta t_2),\,\tilde{u}} &= d_{\theta t_1,(\theta t_2)\,\tilde{u}}\Big(\bigsqcup\{d_{\theta t_2,\tilde{v}} \mid (\theta t_1)\,(\theta t_2)\,\tilde{v} \longrightarrow_D^* (\theta t_2)\tilde{v}\}\Big) \\ &\sqsubseteq [\![t_1]\!](\rho_{t_1 t_2,\tilde{u},\theta})\Big([\![t_2]\!](\rho_{t_1 t_2,\tilde{u},\theta})\Big) \\ &= [\![t_1\, t_2]\!](\rho_{t_1 t_2,\tilde{u},\theta}). \end{aligned}$$

– Case $t_0 = t_1 \sqcap t_2$: Then $\tilde{u} = \epsilon$. We have $\theta t_0 \longrightarrow_D \theta t_1$ and $\theta t_0 \longrightarrow_D \theta t_1$. By the definition of $d_{\theta t_1,\epsilon}$ and the induction hypothesis,

$$\{\mathsf{s}_\star\} = d_{\theta t_1,\epsilon} \sqsubseteq [\![t_1]\!](\rho_{t_1,\epsilon,\theta}).$$

Similarly

$$\{\mathsf{s}_\star\} = d_{\theta t_2,\epsilon} \sqsubseteq [\![t_2]\!](\rho_{t_2,\epsilon,\theta}).$$

Because $\theta t_0 \longrightarrow_D \theta t_1$, we have $\rho_{t_1,\epsilon,\theta} \sqsubseteq \rho_{t_0,\epsilon,\theta}$. Similarly $\rho_{t_2,\epsilon,\theta} \sqsubseteq \rho_{t_0,\epsilon,\theta}$. By monotonicity of interpretation,

$$\{\mathsf{s}_\star\} = d_{\theta t_1,\epsilon} \sqsubseteq [\![t_1]\!](\rho_{t_1,\epsilon,\theta}) \sqsubseteq [\![t_1]\!](\rho_{t_0,\epsilon,\theta})$$

and

$$\{\mathsf{s}_\star\} = d_{\theta t_1,\epsilon} \sqsubseteq [\![t_1]\!](\rho_{t_1,\epsilon,\theta}) \sqsubseteq [\![t_2]\!](\rho_{t_0,\epsilon,\theta}).$$

Hence

$$\{\mathsf{s}_\star\} \sqsubseteq [\![t_1]\!](\rho_{t_0,\epsilon,\theta}) \sqcap [\![t_2]\!](\rho_{t_0,\epsilon,\theta}) = [\![t_0]\!](\rho_{t_0,\epsilon,\theta}).$$

□

The strategy $\mathcal{S}_P$ of $\mathcal{G}_{\mathrm{L}_0,P}$ on $(\{\mathsf{s}_\star\}, \mathbf{main})$ is defined as follows. Each play in the domain of $\mathcal{S}_P$

$$(\{\mathsf{s}_\star\}, \mathbf{main}) \cdot O_1 \cdot (p_1, g_1) \cdot O_2 \cdot \dots \cdot O_n \cdot (p_k, g_k)$$

is associated with a call-sequence

$$\mathbf{main} \overset{m_1}{\leadsto}_D g_1\,\tilde{u}_1 \overset{m_2}{\leadsto}_D g_2\,\tilde{u}_2 \overset{m_3}{\leadsto}_D \dots \overset{m_{k-1}}{\leadsto}_D g_{k-1}\,\tilde{u}_{k-1} \overset{m_k}{\leadsto}_D g_k\,\tilde{u}_k$$

such that $p_j \sqsubseteq d_{g_j,\tilde{u}_j}$ for every $j = 1, 2, \dots, k$. Let ξ_i be the finite sequence over $\{\mathtt{L}, \mathtt{R}\}$ describing the choice made during $g_{i-1}\,\tilde{u}_{i-1} \overset{m_i}{\leadsto}_D g_i\,\tilde{u}_i$ (where $g_0\,\tilde{u}_0 = \mathbf{main}$). The *canonical associated call-sequence* of the play is the minimum one ordered by the lexicographic ordering on $(m_1, \xi_1, m_2, \xi_2, \dots, m_k, \xi_k)$.

Assume that the above call-sequence is canonical. The next step of this reduction sequence is $[\tilde{u}_k/\tilde{x}_k]t$ if $(g_k\,\widetilde{x_k} = t) \in D$. Let $\vartheta = [\tilde{u}_k/\tilde{x}_k]$. In this situation, the strategy $\mathcal{S}_P$ chooses $(\rho_{t,\epsilon,\vartheta}(f_1), \dots, \rho_{t,\epsilon,\vartheta}(f_n))$ as the next node. This is a valid choice, i.e.:

Lemma 31. $p_k \sqsubseteq [\![D(g_k)]\!]([f_1 \mapsto \rho_{t,\epsilon,\vartheta}(f_1), \dots, f_n \mapsto \rho_{t,\epsilon,\vartheta}(f_n)])$.

Proof. By Lemma 30, we have:

$$d_{\vartheta t,\epsilon} = \{\mathsf{s}_\star\} \sqsubseteq [\![t]\!](\rho_{t,\epsilon,\vartheta}) = ([\![\lambda\widetilde{x_k}.t]\!]([f_1 \mapsto \rho_{t,\epsilon,\vartheta}(f_1), \dots, f_n \mapsto \rho_{t,\epsilon,\vartheta}(f_n)]))(\rho_{t,\epsilon,\vartheta}(\widetilde{x_k})).$$

By definition, $d_{g_k,\tilde{u}_k}$ is the least element such that $d_{g_k,\tilde{u}_k}(\rho_{t,\epsilon,\vartheta}(\widetilde{x_k})) = \{\mathsf{s}_\star\}$. Therefore

$$d_{g_k,\tilde{u}_k} \sqsubseteq [\![\lambda\widetilde{x_k}.t]\!]([f_1 \mapsto \rho_{t,\epsilon,\vartheta}(f_1), \dots, f_n \mapsto \rho_{t,\epsilon,\vartheta}(f_n)]).$$

Since $p_k \sqsubseteq d_{g_k,\tilde{u}_k}$, we have

$$p_k \sqsubseteq d_{g_k,\tilde{u}_k} \sqsubseteq [\![\lambda\tilde{x}_k.t]\!]([f_1 \mapsto \rho_{t,\epsilon,\vartheta}(f_1), \dots, f_n \mapsto \rho_{t,\epsilon,\vartheta}(f_n)]).$$

□

Now we have a play

$$(\{\mathsf{s}_\star\}, \mathbf{main}) \cdot O_1 \cdot (p_1, g_1) \cdot O_2 \cdot \dots \cdot O_n \cdot (p_k, g_k) \cdot (\rho_{t,\epsilon,\theta}(f_1), \dots, \rho_{t,\epsilon,\theta}(f_n))$$

associated with the call-sequence

$$\mathbf{main} \overset{m_1}{\rightsquigarrow}_D g_1\, \tilde{u}_1 \overset{m_2}{\rightsquigarrow}_D g_2\, \tilde{u}_2 \overset{m_3}{\rightsquigarrow}_D \dots \overset{m_{k-1}}{\rightsquigarrow}_D g_{k-1}\, \tilde{u}_{k-1} \overset{m_k}{\rightsquigarrow}_D g_k\, \tilde{u}_k$$

Let (p_{k+1}, g_{k+1}) be the next opponent move. By definition of the game,

$$p_{k+1} \sqsubseteq \rho_{t,\epsilon,\theta}(g_{k+1}) = \bigsqcup\{ d_{g_{k+1},\tilde{v}} \mid \theta t \longrightarrow_D^* g_{k+1}\, \tilde{v},\ g_{k+1} \text{ originates from } t \}.$$

This can be more simply written as

$$p_{k+1} \sqsubseteq \rho_{t,\epsilon,\theta}(g_{k+1}) = \bigsqcup\{ d_{g_{k+1},\tilde{v}} \mid g_k\, \widetilde{u_k} \rightsquigarrow g_{k+1}\, \widetilde{v} \}.$$

Since p_{k+1} is a complete prime, $p_{k+1} \sqsubseteq d_{g_{k+1},\tilde{v}}$ for some $\tilde{v}$ with $g_k\, \widetilde{u_k} \rightsquigarrow g_{k+1}\, \widetilde{v}$. Then we have an associated call-sequence

$$\mathbf{main} \overset{m_1}{\rightsquigarrow}_D g_1\, \tilde{u}_1 \overset{m_2}{\rightsquigarrow}_D g_2\, \tilde{u}_2 \overset{m_3}{\rightsquigarrow}_D \dots \overset{m_{k-1}}{\rightsquigarrow}_D g_{k-1}\, \tilde{u}_{k-1} \overset{m_k}{\rightsquigarrow}_D g_k\, \tilde{u}_k \overset{m_{k+1}}{\rightsquigarrow}_D g_{k+1}\, \tilde{u}_{k+1}$$

of

$$(\{\mathsf{s}_\star\}, \mathbf{main}) \cdot O_1 \cdot (p_1, g_1) \cdot O_2 \cdot \dots \cdot O_n \cdot (p_k, g_k) \cdot (\rho_{t,\epsilon,\theta}(f_1), \dots, \rho_{t,\epsilon,\theta}(f_n)) \cdot (p_{k+1}, g_{k+1}).$$

Lemma 32. *$\mathcal{S}_P$ is a winning strategy of $\mathcal{G}_{\mathrm{L}_0,(P,\Omega)}$ on $(\{\mathsf{s}_\star\}, \mathbf{main})$ if every infinite call-sequence of P satisfies the parity condition.*

Proof. The above argument shows that $\mathcal{S}_P$ is a strategy of $\mathcal{G}_{\mathrm{L}_0,(P,\Omega)}$ on $(\{\mathsf{s}_\star\}, \mathbf{main})$. We prove that it is winning. Assume an infinite play

$$(\{\mathsf{s}_\star\}, \mathbf{main}) \cdot v_1 \cdot (p_1, g_1) \cdot v_2 \cdot (p_2, g_2) \cdot \dots$$

that conforms with $\mathcal{S}_P$ and starts from $(\{\mathtt{s}_\star\}, \mathbf{main})$. Then each odd-length prefix is associated with the canonical call-sequence

$$\pi_k \Vdash \mathbf{main} \overset{m_{k,1}}{\rightsquigarrow}_D g_1\,\widetilde{u}_{k,1} \overset{m_{k,2}}{\rightsquigarrow}_D g_2\,\widetilde{u}_{k,2} \overset{m_{k,3}}{\rightsquigarrow}_D \dots \overset{m_{k,k}}{\rightsquigarrow}_D g_k\,\widetilde{u}_{k,k}.$$

Let $\xi_{k,i}$ be the sequence of $\{\mathtt{L}, \mathtt{R}\}$ describing the choice made during $g_{i-1}\,\widetilde{u}_{k,i-1} \overset{m_{k,i}}{\rightsquigarrow}_D g_i\,\widetilde{u}_i$. By the definitions of $\mathcal{S}_P$ and canonical call-sequence (which is the minimum with respect to the lexicographic ordering on $(m_{k,1}, \xi_{k,1}, \dots, m_{k,k}, \xi_{k,k})$), a prefix of the canonical call-sequence is the canonical call-sequence of the prefix. In other words, $k_{k,i} = k_{k',i}$ and $\xi_{k,i} = \xi_{k',i}$ for every $i \geq 1$ and $k, k' \leq i$. Let us write k_i for $k_{i,i} = k_{i+1,i} = \cdots$ and $\widetilde{u}_i$ for $\widetilde{u}_{i,i} = \widetilde{u}_{i+1,i} = \cdots$. Let $\pi = \xi_{1,1}\xi_{2,2}\xi_{3,3}\dots$. Then we have

$$\pi \Vdash \mathbf{main} \overset{k_1}{\rightsquigarrow}_D g_1\,\widetilde{u}_m \overset{k_2}{\rightsquigarrow}_D g_2\,\widetilde{u}_m \overset{k_3}{\rightsquigarrow}_D \dots .$$

Since every infinite call-sequence satisfies the parity condition, the infinite play is P-winning. □

Soundness The proof of soundness (i.e. $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$ implies $\models_{csa} (P, \Omega)$) can be given in a manner similar to the proof of completeness. To show the contraposition, assume that there exists an infinite call-sequence that violates the parity condition. Let π be the choice of this call-sequence. We construct a strategy $\mathcal{S}_{P,\pi}$ of the opposite game $\overline{\mathcal{G}}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$ by inspecting the reduction sequence. Then, by Lemma 25, there is no winning strategy for $\mathcal{G}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$, which implies $\mathtt{L}_0 \not\models \Phi_{(P,\Omega),csa}$ by Theorem 7.

Definition 23 (Opposite parity game for a program). *The parity game $\overline{\mathcal{G}}_{\mathtt{L},(P,\Omega)}$ is defined as follows:*

$$\begin{aligned} V_P &:= \{ (p, f_i) \mid p \in \mathcal{D}_{\mathtt{L},\kappa_i},\ p\text{: complete coprime} \} \\ V_O &:= \{ (d_1, \dots, d_n) \mid d_1 \in \mathcal{D}_{\mathtt{L},\kappa_1}, \dots, d_n \in \mathcal{D}_{\mathtt{L},\kappa_n} \} \\ E &:= \{ ((p, f_i), (d_1, \dots, d_n)) \mid p \sqsupseteq [\![\lambda\tilde{x}.t_i]\!]([f_1 \mapsto d_1, \dots, f_n \mapsto d_n]) \} \\ &\quad \cup \{ ((d_1, \dots, d_n), (p, f_i)) \mid p \sqsupseteq d_i,\ p\text{: complete coprime} \}. \end{aligned}$$

The priority of $(d_1, \dots, d_n)$ is 0 and that of (p, f_i) is $\Omega(f_i) + 1$.

Corollary 3. *$\overline{\mathcal{G}}_{\mathtt{L}_0,(P,\Omega)}$ is isomorphic to $\overline{\mathcal{G}}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$, i.e. there exists a bijection of nodes that preserves the owner and the priority of each node.*

Proof. The bijection on nodes is given by $(d_1, \dots, d_n) \leftrightarrow (d_1, \dots, d_n)$ and $(p, f_i) \leftrightarrow (p, f_i)$. This bijection preserves edges because of Lemma 28. □

Let $\pi_0 \in \{\mathtt{L}, \mathtt{R}\}^\omega$. Given a head occurrence of t of a term reachable from **main** following π_0 (i.e. $(\mathbf{main}; \pi_0) \longrightarrow_D^* (t\,\tilde{u}; \pi)$ for some $\tilde{u}$), we assign an element $\bar{d}_{t,\tilde{u},\pi} \in \mathcal{D}_{\mathtt{L}_0,\kappa}$ where κ is the simple type of t by induction on the order of κ.

- Case $\widetilde{u}$ is empty: Then $t : \star$ and we define $\bar{d}_{t,\epsilon,\pi} = \{\}$.

- Case $\widetilde{u} = u_1 \dots u_k$: Then $t\,\tilde{u} = t\,u_1\ \dots\ u_k$. Note that the order of u_j $(1 \le j \le k)$ is less than that of t. For every $1 \le j \le k$, we define $\bar{e}_j$ as follows.
  - Case $u_j : \mathtt{int}$: Then $\bar{e}_j = [\![u_j]\!]$. (Note that u_j has no free variable and no function symbol.)
  - Case $u_j : \kappa$: Let $\bar{e}_j$ be the element defined by

$$\bar{e}_j = \bigsqcap\{\, \bar{d}_{u_j,\tilde{v},\pi'} \mid (t\,\tilde{u};\pi) \longrightarrow_D^* (u_j\,\tilde{v};\pi')\,\}.$$

    (By the convention, u_j in the right-hand side originates from the j-th argument of t.)

  Then $\bar{d}_{t,\tilde{u},\pi}$ is the function defined by

$$\bar{d}_{t,\tilde{u},\pi}(x_1,\dots,x_k) := \begin{cases}\{\,\} & (\text{if } \bar{x}_j \sqsubseteq \bar{e}_j \text{ for all } j \in \{1,\dots,k\})\\ \{\mathtt{s}_\star\} & (\text{otherwise}).\end{cases}$$

  Note that $\bar{d}_{t,\tilde{u},\pi}$ is the maximum function such that $\bar{d}_{t,\tilde{u},\pi}(\bar{e}_1,\dots,\bar{e}_k) = \{\,\}$.

Let $\theta = [s_1/x_1,\dots,s_k/x_k]$ and assume $(\mathbf{main};\pi_0) \longrightarrow_D^* ((\theta t_0)\,\tilde{u};\pi)$. We define a mapping from function symbols and variables to elements of their semantic domains as follows.

- For a function symbol f_i:

$$\bar{\rho}_{t_0,\tilde{u},\theta,\pi}(f_i) = \bigsqcap\{\, \bar{d}_{f_i,\tilde{v},\pi'} \mid ((\theta t_0)\widetilde{u};\pi) \longrightarrow_D^* (f_i\tilde{v};\pi'),\ f_i \text{ originates from } t_0\,\}$$

- For a variable $x_i : \mathtt{int}$:

$$\bar{\rho}_{t_0,\tilde{u},\theta,\pi}(x_i) = [\![s_i]\!]$$

- For a variable $x_i : \kappa$:

$$\bar{\rho}_{t_0,\tilde{u},\theta,\pi}(x_i) = \bigsqcap\{\, \bar{d}_{s_i,\tilde{v},\pi'} \mid ((\theta t_0)\widetilde{u};\pi) \longrightarrow_D^* (s_i\tilde{v};\pi'),\ s_i \text{ originates from } \theta(x_i)\,\}.$$

Lemma 33. *Let $\theta = [s_1/x_1,\dots,s_k/x_k]$ and assume $(\mathbf{main},\pi_0) \longrightarrow_D^* ((\theta t_0)\,\tilde{u};\pi)$. Let v be an arithmetic expression (i.e. a term of type* `int`*) appearing in t_0. Then $[\![\theta v]\!] = [\![v]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi})$.*

Proof. Similar to the proof of Lemma 29. □

Lemma 34. *Let $\theta = [s_1/x_1,\dots,s_k/x_k]$ and $\pi_0 \in \{\mathtt{L},\mathtt{R}\}^\omega$. Assume that the unique reduction sequence starting from $(\mathbf{main};\pi_0)$ does not terminate and that $(\mathbf{main};\pi_0) \longrightarrow_D^* ((\theta t_0)\,\tilde{u},\pi)$. Then $\bar{d}_{\theta t_0,\tilde{u},\pi} \sqsupseteq [\![t_0]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi})$.*

Proof. By induction on the structure of t_0.

- Case $t_0 = (\,)$: This case never occurs since the reduction sequence does not terminate.
- Case $t_0 = x_i$ for some $1 \le i \le k$: Then $(\theta t_0)\tilde{u} = s_i\,\tilde{u}$. By definition, $\bar{d}_{s_i,\tilde{u},\pi} = \bar{\rho}_{t_0,\tilde{u},\theta,\pi}(x_i)$.

- Case $t_0 = f_i$: Then $(\theta t_0)\tilde{u} = f_i\,\tilde{u}$. Since this occurrence of f_i originates from t_0, we have $\bar{d}_{f_i,\tilde{u},\pi} \sqsupseteq \bar{\rho}_{t_0,\tilde{u},\theta,\pi}(f_i)$.
- Case $t_0 = n$ or t_1 $\texttt{op}$ t_2: Never occurs because the type of t_0 must be of the form $\eta_1 \to \cdots \to \eta_\ell \to \star$.
- Case $t_0 = \textbf{if}\ p(t'_1, \ldots, t'_k)\ \textbf{then}\ t_1\ \textbf{else}\ t_2$: Then $\tilde{u}$ is the empty sequence. Assume that $([\![\theta t'_1]\!], \ldots, [\![\theta t'_k]\!]) \in [\![p]\!]$. The other case can be proved by a similar way. Then we have $(\theta t_0; \pi) \longrightarrow_D (\theta t_1; \pi)$. By definition, $\bar{d}_{\theta t_0,\epsilon;\pi} = \bar{d}_{\theta t_1,\epsilon,\pi} = \{\}$.
  Since every reduction sequence $(\theta t_0; \pi) \longrightarrow^*_D (s\,\tilde{v}; \pi')$ can be factored into $(\theta t_0; \pi) \longrightarrow_D (\theta t_1; \pi) \longrightarrow^*_D (s\,\tilde{v}; \pi')$, we have $\bar{\rho}_{t_0,\epsilon,\theta,\pi} = \bar{\rho}_{t_1,\epsilon,\theta,\pi}$. By the induction hypothesis, $\{\} \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_1,\epsilon,\theta,\pi})$. Hence $\{\} \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi})$.
  By Lemma 33, $[\![\theta t'_j]\!] = [\![t'_j]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi})$ for every $j = 1, \ldots, k$. Hence $([\![\theta t'_1]\!], \ldots, [\![\theta t'_k]\!]) \in [\![p]\!]$ implies $([\![t'_1]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi}), \ldots, [\![t'_k]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi})) \in [\![p]\!]$. By definition, $[\![t_0]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi}) = [\![t_1]\!](\bar{\rho}_{t_0,\tilde{u},\theta,\pi}) \sqsubseteq \{\}$.
- Case $t_0 = \textbf{event}\ a; t$: Then t_0 has type $\star$ and $\tilde{u}$ is the empty sequence. We have $\theta t_0 = \textbf{event}\ a; (\theta t)$. Since every reduction sequence $(\theta t_0; \pi) \longrightarrow^*_D (s\,\tilde{v}; \pi')$ can be factored into $(\theta t_0; \pi) \longrightarrow_D (\theta t; \pi) \longrightarrow^*_D (s\,\tilde{v}; \pi')$, we have $\bar{\rho}_{t_0,\epsilon,\theta,\pi} = \bar{\rho}_{t,\epsilon,\theta,\pi}$. By definition, $\bar{d}_{\theta t_0,\epsilon,\pi} = \bar{d}_{\theta t,\epsilon,\pi} = \{\}$ and $[\![t_0]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi}) = [\![t]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi}) = [\![t]\!](\bar{\rho}_{t,\epsilon,\theta})$. By the induction hypothesis, $\bar{d}_{\theta t,\epsilon,\pi} \sqsupseteq [\![t]\!](\bar{\rho}_{t,\epsilon,\theta,\pi})$. Hence $\bar{d}_{\theta t_0,\epsilon,\pi} \sqsupseteq [\![t_0]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi})$.
- Case $t_0 = t_1\,t_2$: Suppose that $((\theta t_1)(\theta t_2)\tilde{u}; \pi) \longrightarrow^*_D ((\theta t_2)\,\tilde{v}; \pi')$. We first show that $\bar{\rho}_{t_2,\tilde{v},\theta,\pi'}(x) \sqsupseteq \bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}(x)$ for every $x \in \{f_1, \ldots, f_n\} \cup \{x_1, \ldots, x_k\}$.
  - Case $x = x_i : \texttt{int}$: Then $\bar{\rho}_{t_2,\tilde{v},\theta,\pi'}(x_i) = \bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}(x_i) = [\![s_i]\!]$.
  - Case $x = x_i : \kappa$: Then

$$\begin{aligned}\bar{\rho}_{t_2,\tilde{v},\theta,\pi'}(x_i) &= \bigsqcap\{\, \bar{d}_{s_i,\tilde{w},\pi''} \mid ((\theta t_2)\,\tilde{v}; \pi') \longrightarrow^*_D (s_i\tilde{w}; \pi''),\ s_i \text{ originates from } \theta(x_i)\,\} \\ &\sqsupseteq \bigsqcap\{\, \bar{d}_{s_i,\tilde{w},\pi''} \mid ((\theta(t_1\,t_2))\,\tilde{u}; \pi) \longrightarrow^*_D (s_i\tilde{w}; \pi''),\ s_i \text{ originates from } \theta(x_i)\,\} \\ &= \bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}(x_i)\end{aligned}$$

    because $((\theta t_2)\,\tilde{v}; \pi') \longrightarrow^*_D (s_i\tilde{w}; \pi'')$ (where s_i originates from $\theta(x_i)$) implies $((\theta(t_1\,t_2))\,\tilde{u}; \pi) \longrightarrow^*_D ((\theta t_2)\tilde{v}; \pi') \longrightarrow^*_D (s_i\tilde{w}; \pi'')$ (where s_i originates from $\theta(x_i)$).
  - Case $x = f_i$:

$$\begin{aligned}\bar{\rho}_{t_2,\tilde{v},\theta,\pi'}(f_i) &= \bigsqcap\{\, \bar{d}_{f_i,\tilde{w},\pi''} \mid ((\theta t_2)\,\tilde{v}; \pi') \longrightarrow^*_D (f_i\tilde{w}; \pi''),\ f_i \text{ originates from } t_2\,\} \\ &\sqsupseteq \bigsqcap\{\, \bar{d}_{f_i,\tilde{w},\pi''} \mid ((\theta(t_1\,t_2))\,\tilde{u}; \pi) \longrightarrow^*_D (f_i\tilde{w}; \pi''),\ f_i \text{ originates from } t_1 t_2\,\} \\ &= \bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}(f_i)\end{aligned}$$

    because $((\theta t_2)\,\tilde{v}; \pi') \longrightarrow^*_D (f_i\tilde{w}; \pi'')$ (where f_i originates from t_2) implies $((\theta(t_1\,t_2))\,\tilde{u}; \pi) \longrightarrow^*_D ((\theta t_2)\tilde{v}; \pi') \longrightarrow^*_D (f_i\tilde{w}; \pi'')$ (where f_i originates from $t_1 t_2$).

  It is easy to see that $\bar{\rho}_{t_1,(\theta t_2)\,\tilde{u},\theta,\pi} \sqsupseteq \bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}$. By the induction hypothesis,

$$\bar{d}_{\theta t_1,(\theta t_2)\,\tilde{u},\pi} \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_1,(\theta t_2)\,\tilde{u},\theta,\pi}) \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}).$$

By the definition of $\bar{d}_{\theta t_1,(\theta t_2)\,\tilde{u},\pi}$,

$$\bar{d}_{\theta t_1,(\theta t_2)\,\tilde{u},\pi}\Big(\bigsqcap\{\bar{d}_{\theta t_2,\tilde{v},\pi'} \mid ((\theta t_1)\,(\theta t_2)\,\tilde{v};\pi) \longrightarrow^*_D ((\theta t_2)\tilde{v};\pi')\}\Big) = \bar{d}_{(\theta t_1)\,(\theta t_2),\tilde{u},\pi}.$$

By the induction hypothesis, for every reduction sequence $((\theta t_1)\,(\theta t_2)\,\tilde{u};\pi) \longrightarrow^*_D ((\theta t_2)\tilde{v};\pi')$, we have $\bar{d}_{\theta t_2,\tilde{v},\pi'} \sqsupseteq [\![t_2]\!](\bar{\rho}_{t_2,\tilde{v},\theta,\pi'})$. Since $\bar{\rho}_{t_2,\tilde{v},\theta,\pi'}(x) \sqsupseteq \bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi'}(x)$ for every x, we have

$$\bar{d}_{\theta t_2,\tilde{v},\pi'} \sqsupseteq [\![t_2]\!](\bar{\rho}_{t_2,\tilde{v},\theta,\pi'}) \sqsupseteq [\![t_2]\!](\bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}).$$

Because the reduction sequence $((\theta t_1)\,(\theta t_2)\,\tilde{u};\pi) \longrightarrow^*_D ((\theta t_2)\tilde{v};\pi)$ is arbitrary,

$$\Big(\bigsqcap\{\bar{d}_{\theta t_2,\tilde{v},\pi'} \mid ((\theta t_1)\,(\theta t_2)\,\tilde{v};\pi) \longrightarrow^*_D ((\theta t_2)\tilde{v};\pi')\}\Big) \sqsupseteq [\![t_2]\!](\bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}).$$

Therefore, by monotonicity of interpretation,

$$\begin{aligned}\bar{d}_{(\theta t_1)(\theta t_2),\tilde{u},\pi} &= \bar{d}_{\theta t_1,(\theta t_2)\,\tilde{u},\pi}\Big(\bigsqcap\{\bar{d}_{\theta t_2,\tilde{v},\pi'} \mid ((\theta t_1)\,(\theta t_2)\,\tilde{v};\pi) \longrightarrow^*_D ((\theta t_2)\tilde{v};\pi')\}\Big)\\ &\sqsupseteq [\![t_1]\!](\bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi})\Big([\![t_2]\!](\bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi})\Big)\\ &= [\![t_1\,t_2]\!](\bar{\rho}_{t_1 t_2,\tilde{u},\theta,\pi}).\end{aligned}$$

– Case $t_0 = t_1 \Box t_2$: Then $\tilde{u} = \epsilon$. Assume that $\pi = \mathtt{L}\pi'$; the other case can be proved similarly.
Then we have $(\theta t_0;\pi) \longrightarrow_D (\theta t_1;\pi')$. By the definition of $\bar{d}_{\theta t_1,\epsilon,\pi'}$ and the induction hypothesis,

$$\{\,\} = \bar{d}_{\theta t_1,\epsilon,\pi'} \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_1,\epsilon,\theta,\pi'}).$$

Because $(\theta t_0;\pi) \longrightarrow_D (\theta t_1;\pi')$, we have $\bar{\rho}_{t_1,\epsilon,\theta,\pi'} \sqsupseteq \bar{\rho}_{t_0,\epsilon,\theta,\pi}$. By monotonicity of interpretation,

$$\{\,\} = \bar{d}_{\theta t_1,\epsilon,\pi'} \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_1,\epsilon,\theta,\pi'}) \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi'}).$$

Hence

$$\{\,\} \sqsupseteq [\![t_1]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi'}) \sqcap [\![t_2]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi'}) = [\![t_0]\!](\bar{\rho}_{t_0,\epsilon,\theta,\pi'}).$$

□

The strategy $\mathcal{S}_{P,\pi_0}$ of $\mathcal{G}_{\mathrm{L}_0,P}$ on $(\{\mathsf{s}_\star\},\mathbf{main})$ is defined as follows. Each play in the domain of $\mathcal{S}$

$$(\{\mathsf{s}_\star\},\mathbf{main}) \cdot O_1 \cdot (p_1, g_1) \cdot O_2 \cdot \dots \cdot O_n \cdot (p_k, g_k)$$

is associated with a call-sequence

$$(\mathbf{main};\pi_0) \overset{m_1}{\rightsquigarrow}_D (g_1\,\tilde{u}_1;\pi_1) \overset{m_2}{\rightsquigarrow}_D (g_2\,\tilde{u}_2;\pi_2) \overset{m_3}{\rightsquigarrow}_D \dots \overset{m_{k-1}}{\rightsquigarrow}_D (g_{k-1}\,\tilde{u}_{k-1};\pi_{k-1}) \overset{m_k}{\rightsquigarrow}_D (g_k\,\tilde{u}_k;\pi_k)$$

such that $p \sqsupseteq \bar{d}_{g_j,\tilde{u}_j}$ for every $j = 1, 2, \dots, k$. The *canonical associated call-sequence* of the play is the minimum one ordered by the lexicographic ordering of reduction steps $(m_1, m_2, \dots, m_k)$.

Assume that the above call-sequence is canonical. The next step of this reduction sequence is $([\tilde{u}_k/\tilde{x}_k]t;\pi_k)$ if $(g_k\,\widetilde{x_k} = t) \in D$. Let $\vartheta = [\tilde{u}_k/\tilde{x}_k]$. In this situation, the strategy $\mathcal{S}_{P,\pi}$ chooses $(\bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_1), \dots, \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_n))$ as the next node. This is a valid choice, i.e.:

Lemma 35. $p_k \sqsupseteq [\![D(g_k)]\!]([f_1 \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_1), \dots, f_n \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_n)])$.

Proof. By Lemma 34, we have:

$$\bar{d}_{\vartheta t,\epsilon,\pi_k} = \{\} \sqsupseteq [\![t]\!](\bar{\rho}_{t,\epsilon,\vartheta,\pi_k}) = [\![\lambda\widetilde{x_k}.t]\!]([f_1 \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_1), \dots, f_n \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_n)])(\bar{\rho}_{t,\epsilon,\vartheta,\pi}(\widetilde{x_k})).$$

By definition, $\bar{d}_{g_k,\tilde{u}_k,\pi_k}$ is the greatest element such that $\bar{d}_{g_k,\tilde{u}_k,\pi_k}(\bar{\rho}_{t,\epsilon,\vartheta,\pi}(\widetilde{x_k})) = \{\}$. Therefore

$$\bar{d}_{g_k,\tilde{u}_k} \sqsupseteq [\![\lambda\widetilde{x_k}.t]\!]([f_1 \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_1), \dots, f_n \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_n)]).$$

Since $p_k \sqsupseteq \bar{d}_{g_k,\tilde{u}_k,\pi_k}$, we have

$$p_k \sqsupseteq \bar{d}_{g_k,\tilde{u}_k,\pi_k} \sqsupseteq [\![\lambda\tilde{x}_k.t]\!]([f_1 \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_1), \dots, f_n \mapsto \bar{\rho}_{t,\epsilon,\vartheta,\pi_k}(f_n)]).$$

□

Now we have a play

$$(\{\mathsf{s}_\star\}, \mathbf{main}) \cdot O_1 \cdot (p_1, g_1) \cdot O_2 \cdot \dots \cdot O_n \cdot (p_k, g_k) \cdot (\bar{\rho}_{t,\epsilon,\theta,\pi_k}(f_1), \dots, \bar{\rho}_{t,\epsilon,\theta,\pi_k}(f_n))$$

associated with the call sequence

$$(\mathbf{main}; \pi_0) \overset{m_1}{\rightsquigarrow}_D (g_1\, \tilde{u}_1; \pi_1) \overset{m_2}{\rightsquigarrow}_D (g_2\, \tilde{u}_2; \pi_2) \overset{m_3}{\rightsquigarrow}_D \dots \overset{m_{k-1}}{\rightsquigarrow}_D (g_{k-1}\, \tilde{u}_{k-1}; \pi_{k-1}) \overset{m_k}{\rightsquigarrow}_D (g_k\, \tilde{u}_k; \pi_k)$$

Let (p_{k+1}, g_{k+1}) be the next opponent move. By definition of the game,

$$\begin{aligned} p_{k+1} \sqsupseteq \bar{\rho}_{t,\epsilon,\theta}(g_{k+1}) &= \bigsqcap\{\, \bar{d}_{g_{k+1},\tilde{v},\pi} \mid (\theta t; \pi_k) \longrightarrow_P^* (f_{i_{k+1}}\tilde{v}; \pi),\ g_{k+1} \text{ originates from } t \,\} \\ &= \bigsqcap\{\, \bar{d}_{g_{k+1},\tilde{v},\pi} \mid (\theta t; \pi_k) \longrightarrow_P^* (g_k\, \tilde{u}_k; \pi_k) \rightsquigarrow_D (g_{k+1}\, \tilde{v}; \pi) \,\}. \end{aligned}$$

Since p_{k+1} is a complete coprime, $p_{k+1} \sqsupseteq \bar{d}_{g_{k+1},\tilde{v},\pi}$ for some $\tilde{v}$ and π with $(g_k\, \tilde{u}_k; \pi_k) \rightsquigarrow_D (g_{k+1}\, \tilde{v}; \pi)$. Hence we have an associated call-sequence

$$(\mathbf{main}; \pi_0) \overset{m_1}{\rightsquigarrow}_D (g_1\, \tilde{u}_1; \pi_1) \overset{m_2}{\rightsquigarrow}_D (g_2\, \tilde{u}_2; \pi_2) \overset{m_3}{\rightsquigarrow}_D \dots \overset{m_k}{\rightsquigarrow}_D (g_k\, \tilde{u}_k; \pi_k) \overset{m_{k+1}}{\rightsquigarrow} (g_{k+1}\, \tilde{u}; \pi)$$

of

$$(\{\mathsf{s}_\star\}, \mathbf{main}) \cdot O_1 \cdot (p_1, g_1) \cdot O_2 \cdot \dots \cdot O_n \cdot (p_k, g_k) \cdot (\bar{\rho}_{t,\epsilon,\theta,\pi_k}(f_1), \dots, \bar{\rho}_{t,\epsilon,\theta,\pi_k}(f_n)) \cdot (p_{k+1}, g_{k+1})$$

as desired.

Lemma 36. *If the (unique) call sequence following π_0 does not satisfy the parity condition, then $\mathcal{S}_{P,\pi_0}$ is an winning strategy of $\overline{\mathcal{G}}_{\mathrm{L}_0,(P,\Omega)}$ on* $(\{\}, \mathbf{main})$.

Proof. The above argument shows that $\mathcal{S}_{P,\pi_0}$ is a strategy of $\overline{\mathcal{G}}_{\mathrm{L}_0,(P,\Omega)}$ on $(\{\}, \mathbf{main})$. We prove that this is winning.

Assume an infinite play

$$(\{\}, \mathbf{main}) \cdot v_1 \cdot (p_1, g_1) \cdot v_2 \cdot (p_2, g_2) \cdot \dots$$

that conforms with $\mathcal{S}_{P,\pi}$ and starts from $(\{\mathtt{s}_\star\}, \textbf{main})$. Then each odd-length prefix is associated with the canonical call-sequence

$$(\textbf{main}; \pi_0) \overset{k_{m,1}}{\rightsquigarrow}_D (g_1\, \widetilde{u}_{m,1}; \pi_{m,1}) \overset{k_{m,2}}{\rightsquigarrow}_D (g_2\, \widetilde{u}_{m,2}; \pi_{m_2}) \overset{k_{m,3}}{\rightsquigarrow}_D \dots \overset{k_{m,m}}{\rightsquigarrow}_D (g_k\, \widetilde{u}_{m,m}; \pi_{m,m}).$$

By the definitions of $\mathcal{S}_{P,\pi_0}$ and canonical call-sequence (which is the minimum with respect to the lexicographic ordering on $(k_{m,1}, \dots, k_{m,m})$), a prefix of the canonical call-sequence is the canonical call-sequence of the prefix. In other words, $k_{m,i} = k_{m',i}$ for every $i \geq 1$ and $m, m' \geq i$. Since the reduction sequence is completely determined by a choice π_0, the initial term **main** and the number of steps, we have $\pi_{m,i} = \pi_{m',i}$ for every $i \geq 1$ and $m, m' \geq i$. Let us write k_i for $k_{i,i} = k_{i+1,i} = \cdots$, $\widetilde{u}_i$ for $\widetilde{u}_{i,i} = \widetilde{u}_{i+1,i} = \cdots$ and π_i for $\pi_{i,i} = \pi_{i+1,i} = \cdots$. Now we have an infinite call-sequence

$$\pi_0 \Vdash \textbf{main} \overset{k_1}{\rightsquigarrow}_D g_1\, \widetilde{u}_m \overset{k_2}{\rightsquigarrow}_D g_2\, \widetilde{u}_m \overset{k_3}{\rightsquigarrow}_D \dots$$

following π_0. By Corollary 1, this is the unique call-sequence following π_0. Hence, by the assumption, this infinite call-sequence does not satisfy the parity condition. This means that this play is P-winning (recall the definition of the priorities of $\overline{\mathcal{G}}_{\mathtt{L}_0,(P,\Omega)}$). □

Proof (Proof of Theorem 4). We prove the following result:

$\models_{csa} (P, \Omega)$ if and only if $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$.

(⇒) Suppose that $\models_{csa} (P, \Omega)$. Then by Lemma 32, there exists a winning strategy of $\mathcal{G}_{\mathtt{L}_0,(P,\Omega)}$ on $(\{\mathtt{s}_\star\}, \textbf{main})$. Then by Corollary 2, there also exists a winning strategy of $\mathcal{G}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$ on $(\{\mathtt{s}_\star\}, \textbf{main})$. By Theorem 7, this implies $\mathtt{s}_\star \in [\![\Phi_{(P,\Omega),csa}]\!]$. By definition, $\mathtt{L}_0 \models \Phi_{(P,\Omega),csa}$.

(⇐) We prove the contraposition. Suppose that $\models_{csa} (P, \Omega)$ does not hold. Then there exist a choice π_0 and an infinite call-sequence following π_0 that does not satisfy the parity condition. Then by Lemma 36, $\mathcal{S}_{P,\pi_0}$ is a winning strategy of $\overline{\mathcal{G}}_{\mathtt{L}_0,(P,\Omega)}$ on $(\{\}, \textbf{main})$. By Corollary 3, there also exists a winning strategy of $\overline{\mathcal{G}}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$ on $(\{\}, \textbf{main})$. Then by Lemma 25, Opponent wins $\mathcal{G}_{\mathtt{L}_0,\Phi_{(P,\Omega),csa}}$ on $(\{\mathtt{s}_\star\}, \textbf{main})$. Hence by Theorem 7, $\mathtt{s}_\star \notin [\![\Phi_{(P,\Omega),csa}]\!]$. □

E.4 Proof of Theorem 5

Assume a total order on the set of states of the automaton $\mathcal{A}$, fixed in the sequel. Recall (IT-EVENT) rule in Fig. 5:

$$\frac{\delta(q,a) = \{q_1, \dots, q_n\} \qquad \Gamma \uparrow \Omega(q_i) \vdash_{\mathcal{A}} t : q_i \Rightarrow t'_i \quad (\text{for each } i \in \{1, \dots, n\})}{\Gamma \vdash_{\mathcal{A}} (\textbf{event}\ a; t) : q \Rightarrow (\textbf{event}\ a; t'_1 \Box \cdots \Box t'_n)} \quad (\text{IT-EVENT})$$

Since the order of $t'_1, \dots, t'_n$ is not important, we can assume without loss of generality that $q_1 < q_2 < \cdots < q_n$ holds for every instance of the above rule used in a derivation.

Here we prove Theorem 5 for the translation using the following rule IT-EVENT′ instead of IT-EVENT.

$$\frac{\delta(q,a) = \{q_1,\dots,q_n\} \qquad q_1 < q_2 < \cdots < q_n \qquad \Gamma \uparrow \Omega(q_i) \vdash_{\mathcal{A}} t : q_i \Rightarrow t'_i \quad (\text{for each } i \in \{1,\dots,n\})}{\Gamma \vdash_{\mathcal{A}} (\textbf{event}\ a; t) : q \Rightarrow (\textbf{event}\ a; t'_1 \Box \cdots \Box t'_n)} \quad (\text{IT-EVENT}')$$

Notations Given an intersection type $\rho = \bigwedge_{1\le i\le k}(\theta_i, m_i)$ and a variable x, we abbreviate the sequence $x_{\theta_1,m_1}x_{\theta_2,m_2}\dots x_{\theta_k,m_k}$ as $\mathtt{dup}(x,\rho)$. For $\rho = \mathtt{int}$, we write $\mathtt{dup}(x,\rho)$ for $x_{\mathtt{int}}$.

For an intersection type $\rho = \bigwedge_{1\le i\le k}(\theta_i, m_i)$, we write $[x : \rho]$ for the intersection type environment $\{x : (\theta_i, m_i, 0) \mid 1 \le i \le k\}$. If $\rho = \mathtt{int}$, $[x : \rho]$ means $\{x : \mathtt{int}\}$. Similarly, for a top-level environment Ξ, we write $[\Xi]$ to mean $\{x : (\theta, m, 0) \mid x : (\theta, m) \in \Xi\}$.

Given an intersection type environment Γ, we write $\Gamma^\sharp$ for $\{x^\sharp : (\theta, m, m') \mid x : (\theta, m, m') \in \Gamma\} \cup \{x^\sharp : \mathtt{int} \mid x : \mathtt{int} \in \Gamma\}$. Similarly, for a top-level environment Ξ, we write $\Xi^\sharp$ for $\{x^\sharp : (\theta, m) \mid x : (\theta, m) \in \Xi\}$.

For an intersection type environment Γ and a top-level environment Ξ, we write $\Gamma \lhd \Xi$ if $f : (\theta, m) \in \Xi$ for each $(f : (\theta, m, m')) \in \Gamma$.

Modified type-based translation Given programs P and P' with $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega)$, the first step to prove Theorem 5 is to compare reduction sequences of P and P'. There is a little gap between reduction sequences of P and P' since a reduction sequence of P' expresses a reduction sequence of P together with a run of $\mathcal{A}$ over the events generated by the reduction sequence of P. In particular a nondeterministic branch in t' comes from either

- a non-deterministic branch $t_1 \Box t_2$ in P, or
- non-determinism of the transition rule of the automaton $\mathcal{A}$.

To fill the gap, we shall distinguish between the two kinds of non-deterministic branches, by using $\circ$ for the latter.

Formally let us introduce a new binary construct $\circ$ to the syntax of terms. Here we use the convention that $\Box$ and $\circ$ are right associative, i.e. $t_1 \Box t_2 \Box t_3$ (resp. $t_1 \circ t_2 \circ t_3$) means $t_1 \Box (t_2 \Box t_3)$ (resp. $t_1 \circ (t_2 \circ t_3)$). The operational behavior of $\circ$ is the same as that of $\Box$, i.e.,

$$(t_1 \circ t_2;\ \mathtt{L}\pi) \xrightarrow{\epsilon}_D (t_1; \pi) \qquad (t_1 \circ t_2;\ \mathtt{R}\pi) \xrightarrow{\epsilon}_D (t_2; \pi)$$

where $\pi \in \{\mathtt{L}, \mathtt{R}\}^\omega$ is a choice sequence. Hence we have

$$((\textbf{event}\ a; (t_1 \circ \ldots \circ t_n));\ \underbrace{\mathtt{R}\ldots\mathtt{R}}_{i-1}\mathtt{L}\pi) \xrightarrow{a}{}^{*}_{D} (t_i;\ \pi)$$

for $1 \le i < n$ and

$$((\textbf{event}\ a; (t_1 \circ \ldots \circ t_n));\ \underbrace{\mathtt{R}\ldots\mathtt{R}}_{n-1}\pi) \xrightarrow{a}{}^{*}_{D} (t_n;\ \pi).$$

$$\frac{\delta(q,a)=\{q_1,\dots,q_n\} \quad (q_1<q_2<\cdots<q_n) \qquad \Gamma\uparrow\Omega(q_i)\vdash_{\mathcal{A}} t:q_i \Rrightarrow t'_i \quad (\text{for each } i\in\{1,\dots,n\})}{\Gamma\vdash_{\mathcal{A}} (\textbf{event}\ a;t):q \Rrightarrow (\textbf{event}\ a;t'_1\circ\cdots\circ t'_n)} \quad (\text{IT-Event-Alt})$$

Fig. 8. Modified type-based transformation rules. Other rules are obtained by replacing $\Rightarrow$ with $\Rrightarrow$ in the rules in Figure 5 except for (IT-Event).

Definition 24 (Modified type-based transformation). *The* modified type-based transformation judgment $\Gamma\vdash_{\mathcal{A}} t:\theta \Rrightarrow t'$ *is a quadruple where* t *is a term without* $\circ$ *and* t' *is a term possibly having* $\circ$. *The* modified type-based transformation relation *is defined by the rules in Fig. 8. The translation* $\vdash_{\mathcal{A}} P \Rrightarrow (P',\Omega)$ *of programs is defined in the same way as* $\vdash_{\mathcal{A}} P \Rightarrow (P',\Omega)$.

The next lemma establishes the connection between the original and modified transformations. Given a program P' with $\circ$, we write $[\Box/\circ]P'$ for the program obtained by replacing $\circ$ with $\Box$.

Lemma 37. *If* $\Xi\vdash_{\mathcal{A}} P \Rrightarrow (P',\Omega)$, *then* $\Xi\vdash_{\mathcal{A}} P \Rightarrow ([\Box/\circ]P',\Omega)$. *Conversely, if* $\Xi\vdash_{\mathcal{A}} P \Rightarrow (P',\Omega)$, *then there exists* P'' *such that* $\Xi\vdash_{\mathcal{A}} P \Rrightarrow (P'',\Omega)$ *and* $P' = [\Box/\circ]P''$.

Proof. Straightforward induction. □

Suppose that $\Xi\vdash_{\mathcal{A}} P \Rrightarrow (P',\Omega)$. Since P' simulates both P and the automaton $\mathcal{A}$, an infinite reduction sequence of P' induces a pair of an infinite event sequence $\widetilde{\ell}$ and a run R of $\mathcal{A}$ over $\widetilde{\ell}$.

Definition 25 (Induced event sequence and run). *Let* D' *be a function definition possibly containing* $\circ$. *The reduction relation* $(s,\pi,q) \xrightarrow{\widetilde{\ell},R}{}^{N}_{D'} (s',\pi',q')$, *meaning that the* N*-step reduction following* π *from* s *to* s' *generates events* $\widetilde{\ell}$ *associated with a run* qR *ending with* q', *is defined by the following rules.*

$$\frac{(s,\pi)\xrightarrow{\epsilon}{}^{N}_{D'}(s',\pi')}{(s,\pi,q)\xrightarrow{\epsilon,\epsilon}{}^{N}_{D'}(s',\pi',q)}$$

$$\frac{\delta(q,a)=\{q_1,\dots,q_n\} \quad (q_1<q_2<\cdots<q_n) \qquad (s_i,\pi,q_i)\xrightarrow{\widetilde{\ell},R}{}^{N}_{D'}(s',\pi',q')}{(\textbf{event}\ a;(s_1\circ\ldots\circ s_n),\underbrace{\mathtt{R}\ldots\mathtt{R}}_{i-1}\mathtt{L}\pi,q)\xrightarrow{a\widetilde{\ell},q_iR}{}^{N+i+1}_{D'}(s',\pi',q')}$$

$$\frac{\delta(q,a)=\{q_1,\dots,q_n\} \quad (q_1<q_2<\cdots<q_n) \qquad (s_n,\pi,q_n)\xrightarrow{\widetilde{\ell},R}{}^{N}_{D'}(s',\pi',q')}{(\textbf{event}\ a;(s_1\circ\ldots\circ s_n),\underbrace{\mathtt{R}\ldots\mathtt{R}}_{n-1}\pi,q)\xrightarrow{a\widetilde{\ell},q_nR}{}^{N+n}_{D'}(s',\pi',q')}$$

If $(s_0, \pi_0, q_0) \xrightarrow{\widetilde{\ell_1}, R_1 N_1}_{D'} (s_1, \pi_1, q_1) \xrightarrow{\widetilde{\ell_2}, R_2 N_2}_{D'} \ldots$, *we write* $\pi_0 \Vdash (s_0, q_0) \xrightarrow{\widetilde{\ell_1}, R_1 N_1}_{D'} (s_1, q_1) \xrightarrow{\widetilde{\ell_2}, R_2 N_2}_{D'} \ldots$. *If the number of steps (resp.* π*) is not important, we write as* $\pi \Vdash (s, q) \xrightarrow{\widetilde{\ell}, R}{}^{*}_{D'} (s', q')$ *(resp.* $(s, q) \xrightarrow{\widetilde{\ell}, R}{}^{N}_{D'} (s', q')$*). Other notations such as* $(s, q) \xrightarrow{\widetilde{\ell}, R}{}^{*}_{D'} (s', q')$ *are defined similarly.*

It is easy to show that, if $(s, q) \xrightarrow{\widetilde{\ell}, R}{}^{*}_{D'} (s', q')$, then qR is a run of $\mathcal{A}$ over $\widetilde{\ell}$. Obviously $\pi \Vdash (s, q) \xrightarrow{\widetilde{\ell}, R}{}^{*}_{D'} (s', q')$ implies $\pi \Vdash s \xrightarrow{\widetilde{\ell}}{}^{*}_{D'} s'$. As proved later in Lemma 41, the converse also holds: Given state q and $s \xrightarrow{\widetilde{\ell}}{}^{*}_{D'} s'$ (with a mild condition), there exist R and q' such that $(s, q) \xrightarrow{\widetilde{\ell}, R}{}^{*}_{D'} (s', q')$.

Basic properties of the type system and transformation We prove Weakening Lemma and Substitution Lemma.

Lemma 38 (Weakening). *If* $\Gamma \vdash_{\mathcal{A}} t : \theta \Rrightarrow t'$ *and* $\Gamma \subseteq \Gamma'$, *then* $\Gamma' \vdash_{\mathcal{A}} t : \theta \Rrightarrow t'$.

Proof. Straightforward induction on structure of the derivations. We discuss only one case below.

– Case for IT-Event-Alt: Then we have

$$\theta = q \qquad \delta(q, a) = \{q_1, \ldots, q_n\} \qquad t = \mathbf{event}\ a; s \qquad t' = \mathbf{event}\ a; (s'_1 \circ \ldots \circ s'_n)$$
$$\forall i \in \{1, \ldots, n\}. \Gamma \uparrow \Omega(q_i) \vdash_{\mathcal{A}} s : \theta \Rrightarrow s'_i.$$

Since $\Gamma \subseteq \Gamma'$, for each $i \in \{1, \ldots, n\}$, we have $\Gamma \uparrow \Omega(q_i) \subseteq \Gamma' \uparrow \Omega(q_i)$. By the induction hypothesis,

$$\forall i \in \{1, \ldots, n\}. \Gamma' \uparrow \Omega(q_i) \vdash_{\mathcal{A}} s : \theta \Rrightarrow s'_i.$$

By using IT-Event-Alt, we obtain $\Gamma' \vdash_{\mathcal{A}} t : \theta \Rrightarrow t'$.

□

In order to simplify the statement of the substitution lemma, we introduce the following abbreviation. For $\rho = \bigwedge_{1 \le i \le l} (\theta_i, m_i)$, we write

$$\Gamma \vdash_{\mathcal{A}} t : \rho \Rrightarrow \widetilde{s} \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \forall i \in \{1, \ldots, l\}. \Gamma \uparrow m_i \vdash_{\mathcal{A}} t : \theta_i \Rrightarrow s_i.$$

where $\widetilde{s} = s_1 s_2 \ldots s_l$. If $\rho = \mathtt{int}$, the judgment $\Gamma \vdash_{\mathcal{A}} t : \rho \Rrightarrow \widetilde{s}$ has the obvious meaning (in this case, $\widetilde{s}$ is of length 1). By using this notation, the application rule can be simply written as

$$\frac{\Gamma \vdash_{\mathcal{A}} t_1 : \rho \to \theta \Rrightarrow t'_1 \qquad \Gamma \vdash_{\mathcal{A}} t_2 : \rho \Rrightarrow \widetilde{t'_2}}{\Gamma \vdash_{\mathcal{A}} t_1\, t_2 : \theta \Rrightarrow t'_1\, \widetilde{t'_2}}$$

Lemma 39 (Substitution). *Assume that*

$$\Gamma \vdash_{\mathcal{A}} u : \rho \Rrightarrow \widetilde{u'}$$
$$((\Gamma \cup [x : \rho]) \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} t : \theta \Rrightarrow t' \qquad (x \notin dom(\Gamma)).$$

Then

$$(\Gamma \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} [u/x]t : \theta \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x, \rho)]t'.$$

Proof. The proof proceeds by induction on the derivation of $((\Gamma \cup [x : \rho]) \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} t : \theta \Rrightarrow t'$, with case analysis on the last rule used.

– Case for IT-Var: The case where $t \neq x$ is trivial. Assume that $t = x$. The type ρ is either `int` or $\bigwedge_{1 \leq i \leq l}(\theta_i, m_i)$. The former case is easy; we prove the latter case.
  Assume that $\rho = \bigwedge_{1 \leq i \leq l}(\theta_i, m_i)$. Since $((\Gamma \cup [x : \rho]) \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} x : \theta \Rrightarrow t'$, there exists $k \in \{1, \dots, l\}$ such that $\theta = \theta_k$, $t' = x_{\theta_k, m_k}$ and $n = m_k$. By the assumption $\Gamma \vdash_{\mathcal{A}} u : \rho \Rrightarrow \widetilde{u'}$, we have $\Gamma \uparrow m_k \vdash_{\mathcal{A}} u : \theta_k \Rrightarrow u'_k$. Since $[u/x]x = u$ and $[\widetilde{u'}/\mathtt{dup}(x, \rho)]x_{\theta_k, m_k} = u'_k$, we have

$$\Gamma \uparrow m_k \vdash_{\mathcal{A}} [u/x]x : \theta_k \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x, \rho)]x_{\theta_k, m_k}.$$

  By Weakening (Lemma 38), we have

$$(\Gamma \uparrow m_k) \cup \Gamma' \vdash_{\mathcal{A}} [u/x]x : \theta_k \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x, \rho)]x_{\theta_k, m_k}.$$

  as required.
– Case for IT-Event-Alt: Then $t = \textbf{event}\, a; t_1$ and we have

$$\theta = q \qquad \delta(q, a) = \{q_1, \dots, q_k\} \qquad q_1 < q_2 < \cdots < q_k$$
$$t' = \textbf{event}\, a; (t'_1 \circ \ldots \circ t'_k)$$
$$\forall j \in \{1, \dots, k\}.(((\Gamma \cup [x : \rho]) \uparrow n) \cup \Gamma') \uparrow \Omega(q_j) \vdash_{\mathcal{A}} t_1 : q_j \Rrightarrow t'_j.$$

  For each $j \in \{1, \dots, k\}$, we have

$$(((\Gamma \cup [x : \rho]) \uparrow n) \cup \Gamma') \uparrow \Omega(q_j) = ((\Gamma \cup [x : \rho]) \uparrow n \uparrow \Omega(q_j)) \cup (\Gamma' \uparrow \Omega(q_j))$$
$$((\Gamma \uparrow n) \cup \Gamma') \uparrow \Omega(q_j) = (\Gamma \uparrow n \uparrow \Omega(q_j)) \cup (\Gamma' \uparrow \Omega(q_j)).$$

  Thus, one can apply the induction hypotheses, obtaining

$$\forall j \in \{1, \dots, k\}.((\Gamma \uparrow n) \cup \Gamma') \uparrow \Omega(q_j) \vdash_{\mathcal{A}} [u/x]t_1 : q_j \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x, \rho)]t'_j.$$

  By using IT-Event-Alt, we have the following as required:

$$(\Gamma \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} [u/x](\textbf{event}\, a; t_1) : q \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x, \rho)](\textbf{event}\, a; (t'_1 \circ \ldots \circ t'_k)).$$

– Case for IT-App: Then $t = t_1\ t_2$ and we have

$$t' = t'_1\ t'_{2,1} \dots t'_{2,k} \qquad ((\Gamma \cup [x:\rho]) \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} t_1 : \bigwedge_{1\le j\le k} (\theta'_j, m'_j) \to \theta \Rrightarrow t'_1$$
$$\forall j \in \{1,\dots,k\}.(((\Gamma \cup [x:\rho]) \uparrow n) \cup \Gamma') \uparrow m'_j \vdash_{\mathcal{A}} t_2 : \theta'_j \Rrightarrow t'_{2,j}.$$

By the induction hypothesis, we have

$$(\Gamma \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} [u/x]t_1 : \bigwedge_{1\le j\le k} (\theta'_j, m'_j) \to \theta \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x,\rho)]t'_1$$
$$\forall j \in \{1,\dots,k\}.((\Gamma \uparrow n) \cup \Gamma') \uparrow m'_j \vdash_{\mathcal{A}} [u/x]t_2 : \theta'_j \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x,\rho)]t'_{2,j}.$$

By using IT-App, we obtain:

$$(\Gamma \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} [u/x](t_1\ t_2) : \theta \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x,\rho)](t'_1\ t'_{2,1} \dots t'_{2,k})$$

as required.

– Case for IT-Abs: Then $t = \lambda y.s$ and we have

$$\theta = \bigwedge_{1\le j\le k} (\theta'_j, m'_j) \to \theta'_0 \qquad t' = \lambda y_{\theta'_1,m'_1} \dots y_{\theta'_k,m'_k}.s'$$
$$y \notin dom((\Gamma \cup [x:\rho]) \uparrow n) \qquad ((\Gamma \cup [x:\rho]) \uparrow n) \cup \Gamma' \cup [y : \bigwedge_{1\le j\le k} (\theta'_j, m'_j)] \vdash_{\mathcal{A}} s : \theta'_0 \Rrightarrow s'$$

By the induction hypothesis, we have

$$(\Gamma \uparrow n) \cup \Gamma' \cup [y : \bigwedge_{1\le j\le k} (\theta'_j, m'_j)] \vdash_{\mathcal{A}} [u/x]s : \theta'_0 \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x,\rho)]s'.$$

By using IT-Abs, we obtain:

$$(\Gamma \uparrow n) \cup \Gamma' \vdash_{\mathcal{A}} [u/x]\lambda y.s : \bigwedge_{1\le j\le k} (\theta'_j, m'_j) \to \theta'_0 \Rrightarrow [\widetilde{u'}/\mathtt{dup}(x,\rho)]\lambda y_{\theta'_1,m'_1} \dots y_{\theta'_k,m'_k}.s'$$

as required.

□

Simulations in both directions Given $\Xi \vdash_{\mathcal{A}} P \Rrightarrow (P', \Omega)$, we show that the following two data are equivalent:

1. A reduction sequence of P together with a run of $\mathcal{A}$ over the generated event sequence.
2. A reduction sequence of P'.

We first prove the direction (1) ⇒ (2).

Lemma 40. *Assume*

- $\Xi \vdash_{\mathcal{A}} D \Rrightarrow D'$,
- $\Gamma \vdash_{\mathcal{A}} t : q \Rrightarrow s$,
- $\Gamma \lhd \Xi$, *and*
- $t \xrightarrow{\widetilde{\ell}}{}^{*}_{D} t'$.

Let qR be an arbitrary run of $\mathcal{A}$ over $\widetilde{\ell}$ and q' be the last state of qR. Then there exist a term s' and a type environment $\Gamma' \lhd \Xi$ such that

$$(s, q) \xrightarrow{\widetilde{\ell}, R}{}^{*}_{D'} (s', q')$$

and

$$\Gamma' \vdash_{\mathcal{A}} t' : q' \Rrightarrow s'.$$

Proof. By induction on the length of the reduction sequence $t \xrightarrow{\widetilde{\ell}}{}^{*}_{D} t'$. The claim trivially holds if the length is 0; we assume that the length is not 0. The proof proceeds by case analysis on the shape of t.

- Case $t = \textbf{event}\ a; t_1$: Then the reduction sequence $t \xrightarrow{\widetilde{\ell}}{}^{*}_{D} t'$ is of the form

  $$\textbf{event}\ a; t_1 \xrightarrow{a}_{D} t_1 \xrightarrow{\widetilde{\ell'}}{}^{*}_{D} t'$$

  with $\widetilde{\ell} = a\widetilde{\ell'}$. Since the last rule used to derive $\Gamma \vdash_{\mathcal{A}} t : q \Rrightarrow s$ is IT-Event-Alt, we have:

  $$\begin{aligned} \delta(q, a) = \{q_1, \dots, q_k\} \qquad (q_1 < q_2 < \cdots < q_k) \\ \forall i \in \{1, \dots, k\}. \Gamma \uparrow \Omega(q_i) \vdash_{\mathcal{A}} t_1 : q_i \Rrightarrow s_i \\ s = \textbf{event}\ a; (s_1 \circ \ldots \circ s_k). \end{aligned}$$

  Since qR is a run over $\widetilde{\ell} = a\widetilde{\ell'}$, it must be of the form $qR = qq_iR'$ for some $1 \le i \le k$, where q_iR' is a run over $\widetilde{\ell'}$. By applying the induction hypothesis to $t_1 \xrightarrow{\widetilde{\ell'}}{}^{*}_{D} t'$, there exists s' such that

  $$\Gamma' \vdash_{\mathcal{A}} t' : q' \Rrightarrow s' \qquad (s_i, q_i) \xrightarrow{\widetilde{\ell'}, R'}{}^{*}_{D'} (s', q')$$

  for some $\Gamma' \lhd \Xi$. Then we have

  $$(\textbf{event}\ a; (s_1 \circ \ldots \circ s_k), q) \xrightarrow{a, q_i}{}^{*}_{D'} (s_i, q_i) \xrightarrow{\widetilde{\ell'}, R'}{}^{*}_{D'} s'.$$

- Case $t = t_1 \Box t_2$: Suppose that the reduction sequence is $t \xrightarrow{\epsilon}_{D} t_1 \xrightarrow{\widetilde{\ell}}{}^{*}_{D} t'$; the case that $t \xrightarrow{\epsilon}_{D} t_2 \xrightarrow{\widetilde{\ell}}{}^{*}_{D} t'$ can be proved similarly. Since the last rule used to derive $\Gamma \vdash_{\mathcal{A}} t : q \Rrightarrow s$ is IT-NonDet, we have:

  $$s = s_1 \Box s_2 \qquad \Gamma \vdash_{\mathcal{A}} t_1 : q \Rrightarrow s_1 \qquad \Gamma \vdash_{\mathcal{A}} t_2 : q \Rrightarrow s_2$$

By applying the induction hypothesis to $t_1 \xrightarrow{\widetilde{\ell}}{}^*_D t'$, we have s' such that

$$\Gamma' \vdash_{\mathcal{A}} t' : q' \Rrightarrow s' \qquad (s_1, q) \xrightarrow{\widetilde{\ell},R}{}^*_{D'} (s', q')$$

for some $\Gamma' \lhd \Xi$. Then

$$(s_1 \Box s_2, q) \xrightarrow{\epsilon,\epsilon}{}^*_{D'} (s_1, q) \xrightarrow{\widetilde{\ell},R}{}^*_{D'} (s', q').$$

- Case $t = \textbf{if } p(t_1, \dots, t_n) \textbf{ then } t_{n+1} \textbf{ else } t_{n+2}$: Suppose $([\![t_1]\!], \dots, [\![t_n]\!]) \in [\![p]\!]$. Then the reduction sequence is $t \xrightarrow{\epsilon}_D t_{n+1} \xrightarrow{\widetilde{\ell}}{}^*_D t'$. Since the last rule used to derive $\Gamma \vdash_{\mathcal{A}} t : q \Rrightarrow s$ is IT-IF, we have:

$$s = \textbf{if } p(s_1, \dots, s_n) \textbf{ then } s_{n+1} \textbf{ else } s_{n+2} \qquad \forall i \in \{1, \dots, n\}. \Gamma \vdash_{\mathcal{A}} t_i : \mathtt{int} \Rrightarrow s_i$$
$$\Gamma \vdash_{\mathcal{A}} t_{n+1} : q \Rrightarrow s_{n+1} \qquad \Gamma \vdash_{\mathcal{A}} t_{n+2} : q \Rrightarrow s_{n+2}$$

By the well-typedness of the program, for every $i \in \{1, \dots, n\}$, t_i consists of only integers and integer operations. Thus, we have $[\![t_i]\!] = [\![s_i]\!]$ for each $i \in \{1, \dots, n\}$ and $s \xrightarrow{\epsilon}_{D'} s_{n+1}$.
By applying the induction hypothesis to $t_{n+1} \xrightarrow{\widetilde{\ell}}{}^*_D t'$, we have

$$\Gamma' \vdash_{\mathcal{A}} t' : q' \Rrightarrow s' \qquad (s_{n+1}, q) \xrightarrow{\widetilde{\ell},R}{}^*_{D'} (s', q')$$

for some $\Gamma' \lhd \Xi$ and s'. Then

$$(s, q) \xrightarrow{\epsilon,\epsilon}{}^*_{D'} (s_{n+1}, q) \xrightarrow{\widetilde{\ell},R}{}^*_{D'} (s', q').$$

The case where $([\![t_1]\!], \dots, [\![t_n]\!]) \notin [\![p]\!]$ is similar.
- Case $t = f\ t_1 \dots t_n$: Suppose $D(f) = \lambda x_1 \dots x_n.t_0$. In this case, the reduction sequence is $t \xrightarrow{\epsilon}_D [t_1/x_1] \dots [t_n/x_n]t_0 \xrightarrow{\widetilde{\ell}}{}^*_D t'$. Since $\Gamma \vdash_{\mathcal{A}} t : q \Rrightarrow s$, we have $f : (\theta, m, m) \in \Gamma$ such that

$$\theta = \rho_1 \to \dots \rho_n \to q$$
$$\forall i \in \{1, \dots, n\}.\ \Gamma \vdash_{\mathcal{A}} t_i : \rho_i \Rrightarrow \widetilde{s_i}$$
$$s = f_{\theta,m}\, \widetilde{s_1}\ \dots\ \widetilde{s_n}.$$

Since $\Xi \vdash_{\mathcal{A}} D \Rrightarrow D'$, we have

$$D'(f_{\theta,m}) = \lambda \mathsf{dup}(x_1, \rho_1) \dots \mathsf{dup}(x_n, \rho_n).s_0$$
$$[\Xi] \cup [x_1 : \rho_1] \cup \dots \cup [x_n : \rho_n] \vdash_{\mathcal{A}} t_0 : q \Rrightarrow s_0$$

By using Weakening (Lemma 38), we have

$$\forall i \in \{1, \dots, n\}.\ [\Xi] \cup \Gamma \vdash_{\mathcal{A}} t_i : \rho_i \Rrightarrow \widetilde{s_i}$$
$$[\Xi] \cup \Gamma \cup [x_1 : \rho_1] \cup \dots \cup [x_n : \rho_n] \vdash_{\mathcal{A}} t_0 : q \Rrightarrow s_0$$

Since

$$([\Xi] \cup \Gamma \cup [x_1 : \rho_1] \cup \cdots \cup [x_n : \rho_n]) \uparrow 0 = [\Xi] \cup \Gamma \cup [x_1 : \rho_1] \cup \cdots \cup [x_n : \rho_n],$$

by using Substitution Lemma 39 repeatedly, we have

$$[\Xi] \cup \Gamma \vdash_{\mathcal{A}} [t_1/x_1] \ldots [t_n/x_n]t_0 : q \Rrightarrow [\widetilde{s_1}/\mathtt{dup}(x_1, \rho_1)] \ldots [\widetilde{s_n}/\mathtt{dup}(x_n, \rho_n)]s_0.$$

Since $([\Xi] \cup \Gamma) \lhd \Xi$, we can apply the induction hypothesis to $[t_1/x_1] \ldots [t_n/x_n]t_0 \xrightarrow{\widetilde{\ell}}{}^*_D t'$; we have

$$\Gamma' \vdash_{\mathcal{A}} t' : q' \Rrightarrow s' \qquad (s'', q) \xrightarrow{\widetilde{\ell},R}{}^{N'}_{D'} (s', q')$$

for some $\Gamma' \lhd \Xi$ and s', where

$$s'' = [\widetilde{s_1}/\mathtt{dup}(x_1, \rho_1)] \ldots [\widetilde{s_n}/\mathtt{dup}(x_n, \rho_n)]s_0.$$

We have $(s, q) \xrightarrow{\epsilon,\epsilon}{}^*_{D'} (s'', q) \xrightarrow{\widetilde{\ell},R}{}^*_{D'} (s', q')$ as desired.

□

An infinite analogue of this lemma can be obtained as a corollary.

Corollary 4. *Assume*

$$\Xi \vdash_{\mathcal{A}} D \Rrightarrow D' \qquad \Gamma \vdash_{\mathcal{A}} t_0 : q_0 \Rrightarrow s_0 \qquad \Gamma \lhd \Xi$$

and an infinite reduction sequence

$$t_0 \xrightarrow{\ell_1}_D t_1 \xrightarrow{\ell_1}_D t_2 \xrightarrow{\ell_2}_D \cdots$$

generating an infinite event sequence $\widetilde{\ell} = \ell_1\ell_2 \cdots$. *Let* q_0R *be an infinite run of* $\mathcal{A}$ *over the infinite sequence* $\widetilde{\ell}$. *Then there exist* $\{(q_i, R_i, s_i)\}_{i\in\omega}$ *such that*

$$(s_0, q_0) \xrightarrow{\ell_1,R_1}{}^*_{D'} (s_1, q_1) \xrightarrow{\ell_2,R_2}{}^*_{D'} (s_2, q_2) \xrightarrow{\ell_3,R_3}{}^*_{D'} \cdots$$

and $R = R_1R_2 \ldots$.

Proof. By using Lemma 40, one can construct by induction on $i > 0$ a family $\{(q_i, R_i, s_i, \Gamma_i)\}_{i\in\omega}$ that satisfies

$$(s_{i-1}, q_{i-1}) \xrightarrow{\ell_i,R_i}{}^*_{D'} (s_i, q_i) \qquad \Gamma_i \lhd \Xi \qquad \Gamma_i \vdash_{\mathcal{A}} t_i : q_i \Rrightarrow s_i \qquad R_1R_2 \ldots R_i \text{ is a prefix of } R.$$

Since the length of $R_1R_2 \ldots R_i$ is equivalent to that of $\ell_1\ell_2 \ldots \ell_i$, the sequence $R_1R_2 \ldots$ is indeed an infinite sequence and thus equivalent to R. □

We show the converse in a bit stronger form.

Lemma 41. *Assume*

$$\Xi \vdash_{\mathcal{A}} D \Rrightarrow D' \qquad \Gamma_1 \lhd \Xi \qquad \Gamma_2 \lhd \Xi^\sharp \qquad \Gamma_1 \cup \Gamma_2 \vdash_{\mathcal{A}} t : q \Rrightarrow s \qquad (s,\pi) \xrightarrow{\widetilde{\ell}}{}^{N}_{D'^\sharp} (s',\pi')$$

where t does not contain λ-abstraction and s' is not of the form $s'_1 \circ s'_2$. Then there exist a run qR over $\widetilde{\ell}$ and a state q' such that

$$(s,\pi,q) \xrightarrow{\widetilde{\ell},R}{}^{N}_{D'^\sharp} (s',\pi',q')$$

Furthermore there exist a term t' and a type environment Γ'_1 such that

$$\Gamma'_1 \lhd \Xi \qquad \Gamma'_1 \cup (\Gamma_2 \uparrow \mathbf{max}(\Omega(R))) \vdash_{\mathcal{A}} t' : q' \Rrightarrow s' \qquad t \xrightarrow{\widetilde{\ell}}{}^{*}_{D^\sharp} t'.$$

Proof. By induction on the length of the reduction sequence. The claim trivially holds if the length is 0. Assume that the length is greater than 0.

The proof proceeds by case analysis on the shape of s. By the assumption $s \xrightarrow{\widetilde{\ell}}{}^{*}_{D'^\sharp} s'$, it suffices to consider only the cases where the shape of s matches the lefthand side of a transition rule.

- Case $s = \mathbf{event}\ a; s_0$: By the shape of s, the last rule used on the derivation of $\Gamma \cup \Gamma' \vdash_{\mathcal{A}} t : q \Rrightarrow s$ is either IT-Event-Alt or IT-App (IT-Abs is not applicable since t is assumed to have no abstraction). By induction on the derivation, one can prove that $t = (\mathbf{event}\ a; t_0)\, t_1\, \ldots\, t_k$ for some k. By the simple-type system, one cannot apply $\mathbf{event}\, a; t_0$ to a term; hence $k = 0$ (i.e. $t = \mathbf{event}\, a; t_0$) and the last rule used on the derivation is IT-Event-Alt. Then we have:

$$\delta(q,a) = \{q_1,\ldots,q_n\}\ (q_1 < q_2 < \cdots < q_n)$$
$$\forall i \in \{1,\ldots,n\}.(\Gamma_1 \cup \Gamma_2) \uparrow \Omega(q_i) \vdash_{\mathcal{A}} t_0 : q_i \Rrightarrow s_i$$
$$s_0 = (s_1 \circ \ldots \circ s_l)$$

  Recall that s' is not of the form $s'_1 \circ s'_2$; hence the reduction sequence

$$(s,\pi) \xrightarrow{\widetilde{\ell}}{}^{N}_{D'^\sharp} (s',\pi')$$

  must be of the form

$$(s,\pi) \xrightarrow{a}{}^{k}_{D'^\sharp} (s_i,\pi'') \xrightarrow{\widetilde{\ell'}}{}^{N-k}_{D'^\sharp} (s',\pi')$$

  where $k = i+1$ if $i < n$ and $k = i$ if $i = n$. By the induction hypothesis, there exist a run q_iR' over ℓ', a term t' and a type environment Γ' with $\Gamma'_1 \lhd \Xi$ such that

$$(s_i,\pi'',q_i) \xrightarrow{\widetilde{\ell'},R'}{}^{N-k}_{D'^\sharp} (s',\pi',q')$$

  and

$$\Gamma'_1 \cup (\Gamma_2 \uparrow \Omega_{\mathcal{A}}(q_i) \uparrow \mathbf{max}(\Omega_{\mathcal{A}}(R'))) \vdash_{\mathcal{A}} t' : q' \Rrightarrow s'.$$

  We have $(\Gamma_2 \uparrow \Omega_{\mathcal{A}}(q_i) \uparrow \mathbf{max}(\Omega_{\mathcal{A}}(R'))) = (\Gamma_2 \uparrow \mathbf{max}(\Omega_{\mathcal{A}}(q_iR')))$ and $(s,\pi,q) \xrightarrow{a,q_i}{}^{k}_{D'^\sharp} (s_i,\pi'',q_i) \xrightarrow{\widetilde{\ell'},R'}{}^{N-k}_{D'^\sharp} (s',\pi',q')$. By construction, qq_iR' is a run over $a\widetilde{\ell'}$.

– Case $s = s_1 \Box s_2$: As with the previous case, the last rule used on the derivation is IT-NonDet. Then we have

$$(\Gamma_1 \cup \Gamma_2) \vdash_{\mathcal{A}} t_1 : q \Rrightarrow s_1 \qquad (\Gamma_1 \cup \Gamma_2) \vdash_{\mathcal{A}} t_2 : q \Rrightarrow s_2 \qquad t = t_1 \Box t_2$$

Suppose $\pi = \mathtt{L}\pi''$; the other case can be proved by a similar way. Then the reduction sequence $(s, \pi) \xrightarrow{\widetilde{\ell}}{}^{N}_{D'^\sharp} (s', \pi')$ must be of the form

$$(s, \pi) \xrightarrow{\epsilon}_{D'^\sharp} (s_1, \pi'') \xrightarrow{\widetilde{\ell}}{}^{N-1}_{D'^\sharp} (s', \pi').$$

Hence, by applying the induction hypothesis to $\Gamma_1 \cup \Gamma_2 \vdash_{\mathcal{A}} t_1 : q \Rrightarrow s_1$, we obtain the desired result.

– Case $s = \textbf{if}\ p(s_1, \dots, s_n)\ \textbf{then}\ s_{n+1}\ \textbf{else}\ s_{n+2}$: As with the previous cases, the last rule used on the derivation is IT-If. Hence we have

$$\begin{aligned}
&t = \textbf{if}\ p(t_1, \dots, t_n)\ \textbf{then}\ t_{n+1}\ \textbf{else}\ t_{n+2}\\
&\forall i \in \{1, \dots, n\}. \Gamma_1 \cup \Gamma_2 \vdash_{\mathcal{A}} t_i : \mathtt{int} \Rrightarrow s_i\\
&\Gamma_1 \cup \Gamma_2 \vdash_{\mathcal{A}} t_{n+1} : q \Rrightarrow s_{n+1} \qquad \Gamma_1 \cup \Gamma_2 \vdash_{\mathcal{A}} t_{n+2} : q \Rrightarrow s_{n+2}
\end{aligned}$$

Recall that the result type of a function cannot be the integer type in our language. This implies that t_i ($1 \le i \le n$) consists only of constants and numerical operations, and thus $t_i = s_i$ for every $1 \le i \le n$.
Suppose that $([\![s_1]\!], \dots, [\![s_n]\!]) = ([\![t_1]\!], \dots, [\![t_n]\!]) \in [\![p]\!]$. Then the reduction sequence $(s, \pi) \xrightarrow{\widetilde{\ell}}{}^{N}_{D'^\sharp} (s', \pi')$ must be of the form

$$(s, \pi) \xrightarrow{\epsilon}_{D'^\sharp} (s_{n+1}, \pi) \xrightarrow{\widetilde{\ell}}{}^{N-1}_{D'^\sharp} (s', \pi').$$

We obtain the desired result by applying the induction hypothesis to $\Gamma_1 \cup \Gamma_2 \vdash_{\mathcal{A}} t_{n+1} : q \Rrightarrow s_{n+1}$; the corresponding reduction sequence is $t \xrightarrow{\epsilon}_{D^\sharp} t_{n+1} \xrightarrow{\widetilde{\ell}}{}^{*}_{D^\sharp} t'$. The case that $([\![s_1]\!], \dots, [\![s_n]\!]) \notin [\![p]\!]$ can be proved similarly.

– Case $s = g\ s_1 \dots s_n$ with $g \in dom(D')$: By the shape of s, the last rule used on the derivation is IT-App or IT-AppInt. By induction on the derivation, we have $t = f t_1 \dots t_k$ for some k. Then $s = f_{\theta,m}\, \widetilde{s'_1}\ \dots\ \widetilde{s'_k}$ and

$$\begin{aligned}
f : (\theta, m, m) &\in \Gamma_1\\
\theta = \rho_1 &\to \dots \rho_k \to q\\
\forall i \in \{1, \dots, k\}.\ \Gamma_1 \cup \Gamma_2 &\vdash_{\mathcal{A}} t_i : \rho_i \Rrightarrow \widetilde{s'_i}.
\end{aligned}$$

Assume that $D(f) = \lambda x_1 \dots x_k.t_{body}$ and $D'(f_{\theta,m}) = \lambda y_1 \dots y_n.s_{body}$. Since $\Xi \vdash_{\mathcal{A}} D \Rrightarrow D'$ and $f : (\theta, m, m) \in \Gamma_1 \lhd \Xi$, we have

$$[\Xi] \vdash_{\mathcal{A}} \lambda x_1 \dots x_k.t_{body} : \theta \Rrightarrow \lambda y_1 \dots y_n.s_{body}.$$

Hence $y_1 \dots y_n = \mathtt{dup}(x_1, \rho_1) \dots \mathtt{dup}(x_k, \rho_k)$ and

$$[\Xi] \cup [x_1 : \rho_1] \cup \dots \cup [x_k : \rho_k] \vdash_{\mathcal{A}} t_{body} : q \Rrightarrow s_{body}.$$

By using Weakening (Lemma 38) and Substitution Lemma (Lemma 39) repeatedly, we have

$[\Xi] \cup \Gamma \cup \Gamma' \vdash_{\mathcal{A}} [t_1/x_1] \dots [t_k/x_k] t_{body} : q \Rrightarrow [\widetilde{u_1}/\mathtt{dup}(x_1, \rho_1)] \dots [\widetilde{u_k}/\mathtt{dup}(x_k, \rho_k)] s_{body}$.

The reduction sequence $(s, \pi) \xrightarrow{\widetilde{\ell}}{}^{N}_{D'^\sharp} (s', \pi')$ must be of the form

$$(s, \pi) \xrightarrow{\epsilon}_{D'^\sharp} ([\widetilde{u_1}/\mathtt{dup}(x_1, \rho_1)] \dots [\widetilde{u_k}/\mathtt{dup}(x_k, \rho_k)] s_{body}, \pi) \xrightarrow{\widetilde{\ell}}{}^{N-1}_{D'^\sharp} (s', \pi').$$

By applying the induction hypothesis to the above judgment, we complete the proof; the corresponding reduction sequence is $t \xrightarrow{\epsilon}_{D^\sharp} [t_1/x_1] \dots [t_k/x_k] t_{body} \xrightarrow{\widetilde{\ell}}{}^{*}_{D^\sharp} t'$ where t' is the term obtained by the induction hypothesis.

– Case $s = g^\sharp\ s_1 \dots s_n$ with $g \in dom(D')$: Similar to the above case.

□

Corollary 5. *Assume*

$$\Xi \vdash_{\mathcal{A}} D \Rrightarrow D' \qquad \Gamma \lhd \Xi \qquad \Gamma \vdash_{\mathcal{A}} t_0 : q_0 \Rrightarrow s_0$$

and an infinite reduction sequence

$$(s_0, \pi_0) \xrightarrow{\ell_1}{}^{N_1}_{D'} (s_1, \pi_1) \xrightarrow{\ell_2}{}^{N_2}_{D'} (s_2, \pi_2) \xrightarrow{\ell_3}{}^{N_3}_{D'} \cdots .$$

Suppose that, for every i, s_i is not of the form $s_{i1} \circ s_{i2}$. Then there exist $\{(q_i, R_i, t_i)\}_{i\in\omega}$ such that

$$(s_0, \pi_0, q_0) \xrightarrow{\ell_1, R_1}{}^{N_1}_{D'} (s_1, \pi_1, q_1) \xrightarrow{\ell_2, R_2}{}^{N_2}_{D'} (s_2, \pi_2, q_2) \xrightarrow{\ell_3, R_3}{}^{N_3}_{D'} \cdots$$

and

$$t_0 \xrightarrow{\ell_1}{}^{*}_{D} t_1 \xrightarrow{\ell_2}{}^{*}_{D} t_2 \xrightarrow{\ell_3}{}^{*}_{D} \cdots .$$

Furthermore $q_0 R_1 R_2 \dots$ is an infinite run of $\mathcal{A}$ over $\ell_1 \ell_2 \dots$.

Proof. By using Lemma 41, one can define a family $\{(q_i, R_i, t_i, \Gamma_i)\}_{i\in\omega}$ that satisfies

$$\begin{aligned} (s_{i-1}, \pi_{i-1}, q_{i-1}) \xrightarrow{\ell_i, R_i}{}^{N_i}_{D'^\sharp} (s_i, \pi_i, q_i) \\ \Gamma_i \lhd \Xi \\ \Gamma_i \vdash_{\mathcal{A}} t_i : q_i \Rrightarrow s_i \\ t_{i-1} \xrightarrow{\widetilde{\ell}}{}^{*}_{D^\sharp} t_i \end{aligned}$$

by induction on $i > 0$. Since t_0 and s_0 do not contain marked symbols as well as the bodies of function definitions in D and D', t_i and s_i dose not have marked symbols for every i. Hence $(s_{i-1}, \pi_{i-1}, q_{i-1}) \xrightarrow{\ell_i, R_i}{}^{N_i}_{D'} (s_i, \pi_i, q_i)$ and $t_{i-1} \xrightarrow{\widetilde{\ell}}{}^{*}_{D} t_i$ for every $i > 0$. Furthermore $q_0 R_1 \dots R_i$ is a run over $\ell_1 \dots \ell_i$ for every i. Hence the infinite sequence $q_0 R_1 R_2 \dots$ is an infinite run over $\ell_1 \ell_2 \dots$. So $\{(q_i, R_i, t_i)\}_{i\in\omega}$ satisfies the requirements. □

Lemma 42. *Assume that* $\Xi \vdash_{\mathcal{A}} P \Rrightarrow (P', \Omega)$. *Let* $P = (\mathbf{main}, D)$ *and* $P' = (\mathbf{main}', D')$. *The following conditions are equivalent.*

1. $\exists \widetilde{\ell} \in \mathbf{InfTraces}(P). \exists R$: *run of* $\mathcal{A}$ *over* $\widetilde{\ell}$. $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R))$ *is* even.
2. *There exist* π *and an infinite reduction sequence*

$$\pi \Vdash (\mathbf{main}', q_I) = (s_0, q_0) \xrightarrow{\ell_1, R_1}{}^{*}_{D'} (s_1, q_1) \xrightarrow{\ell_2, R_2}{}^{*}_{D'} \cdots$$

such that $R_1 R_2 \cdots$ *is an infinite sequence and* $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \cdots))$ *is even.*

Proof. ((1) ⇒ (2)) Assume an infinite reduction sequence

$$\pi \Vdash \mathbf{main} = t_0 \xrightarrow{\ell_1}_D t_1 \xrightarrow{\ell_2}_D t_2 \xrightarrow{\ell_3}_D \cdots$$

and an infinite run $q_I R$ of $\mathcal{A}$ over $\ell_1 \ell_2 \ldots$ (here we implicitly assume that $\ell_1 \ell_2 \ldots$ is an infinite sequence). Since $\Xi \vdash_{\mathcal{A}} P \Rrightarrow (P', \Omega)$, we have $[\Xi] \vdash_{\mathcal{A}} \mathbf{main} : q_I \Rrightarrow \mathbf{main}'$. Then by Corollary 4, there exist $\{(q_i, R_i, s_i)\}_{i \in \omega}$ such that

$$(\mathbf{main}', q_I) = (s_0, q_0) \xrightarrow{\ell_1, R_1}{}^{*}_{D'} (s_1, q_1) \xrightarrow{\ell_2, R_2}{}^{*}_{D'} (s_2, q_2) \xrightarrow{\ell_3, R_3}{}^{*}_{D'} \cdots$$

and $R = R_1 R_2 \ldots$. Then $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \cdots)) = \mathbf{maxInf}(\Omega_{\mathcal{A}}(q_I R_1 R_2 \cdots)) = \mathbf{maxInf}(\Omega_{\mathcal{A}}(R))$ is even by the assumption.

((2) ⇒ (1)) Assume an infinite reduction sequence

$$\pi \Vdash (\mathbf{main}', q_I) = (s_0, q_0) \xrightarrow{\ell_1, R_1}{}^{*}_{D'} (s_1, q_1) \xrightarrow{\ell_2, R_2}{}^{*}_{D'} \cdots$$

such that $R_1 R_2 \cdots$ is an infinite sequence and $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \cdots))$ is even. By Corollary 5, there exists $\{t_i\}_i$ such that

$$\mathbf{main} \xrightarrow{\ell_1}{}^{*}_{D} t_1 \xrightarrow{\ell_2}{}^{*}_{D} \cdots .$$

By construction, $R = q_I R_1 R_2 \cdots$ is an infinite run over $\ell_1 \ell_2 \cdots$. Since R is infinite by the assumption, $\ell_1 \ell_2 \cdots$ is also infinite. Then $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R)) = \mathbf{maxInf}(\Omega_{\mathcal{A}}(q_I R_1 R_2 \cdots)) = \mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \cdots))$ is even by the assumption. □

Lemma 43. *Assume that*

$$\Xi \vdash_{\mathcal{A}} D \Rrightarrow D' \qquad \Gamma_0 \lhd \Xi \qquad \Gamma_0 \vdash_{\mathcal{A}} t : q \Rrightarrow g\,\widetilde{s} \qquad (g\,\widetilde{s}; \pi) \overset{N}{\rightsquigarrow}_{D'} (h_{\theta,m}\,\widetilde{u}; \pi')$$

and $(g\,\widetilde{s}, \pi, q) \xrightarrow{\widetilde{\ell}, R}{}^{N}_{D'} (h_{\theta,m}\,\widetilde{u}, \pi', q')$. *Then* $m = \mathbf{max}(\Omega(R))$.

Proof. Since $\Gamma_0 \vdash_{\mathcal{A}} t : q \Rrightarrow g\,\widetilde{s}$, there exist θ_0 and m_0 such that

$$\begin{aligned}
&t = f\, t_1 \ \ldots\ t_n \\
&\widetilde{s} = \widetilde{s_1} \ \ldots\ \widetilde{s_n} \\
&f : (\theta_0, m_0, m_0) \in \Gamma_0 \\
&\theta_0 = \rho_1 \to \cdots \to \rho_n \to q \\
&\forall i \in \{1, \ldots, n\}.\ \Gamma_0 \vdash_{\mathcal{A}} t_i : \rho_i \Rrightarrow \widetilde{s_i}.
\end{aligned}$$

Suppose $D(f) = \lambda x_1 \dots x_n.t_0$. Since $\Xi \vdash_{\mathcal{A}} D \Rrightarrow D'$, we have

$$D'(f) = \lambda \widetilde{x_1} \dots \widetilde{x_n}.s_0 \qquad [\Xi] \cup [x_1 : \rho_1] \cup \dots \cup [x_n : \rho_n] \vdash_{\mathcal{A}} t_0 : q \Rrightarrow s_0.$$

By easy induction on the structure of the derivation, we have

$$[\Xi] \cup [\Xi^\sharp] \cup [x_1 : \rho_1] \cup \dots \cup [x_n : \rho_n] \vdash_{\mathcal{A}} t_0^\sharp : q \Rrightarrow s_0^\sharp.$$

By using Weakening (Lemma 38) and Substitution Lemma (Lemma 39), we have

$$[\Xi] \cup [\Xi^\sharp] \cup \Gamma_0 \vdash_{\mathcal{A}} [s_1/x_1] \dots [s_n/x_n] t_0^\sharp : q \Rrightarrow [\widetilde{s_1}/\widetilde{x_1}] \dots [\widetilde{s_s}/\widetilde{x_n}] s_0^\sharp.$$

Now, by the definition of call sequence, we have

$$([s_1/x_1] \dots [s_n/x_n] s_0^\sharp, \pi) \xrightarrow{\widetilde{\ell}}{}^{N-1}_{D'^\sharp} (h^\sharp_{\theta,m}\, \widetilde{u'}, \pi').$$

By Lemma 41, there exist R', t' and $\Gamma' \lhd \Xi$ such that

$$([s_1/x_1] \dots [s_n/x_n] s_0^\sharp, \pi, q) \xrightarrow{\widetilde{\ell}, R'}{}^{N-1}_{D'^\sharp} (h^\sharp_{\theta,m}\, \widetilde{u'}, \pi', q')$$

and

$$\Gamma' \cup ([\Xi^\sharp] \uparrow \mathbf{max}(\Omega_{\mathcal{A}}(R'))) \vdash_{\mathcal{A}} t' : q' \Rrightarrow h^\sharp_{\theta,m}\, \widetilde{u'}$$

where q' is the last state in R'.

By $(g\, \widetilde{s}, \pi, q) \xrightarrow{\widetilde{\ell}, R}{}^{N}_{D'} (h_{\theta,m}\, \widetilde{u}, \pi', q')$, we have

$$(g\, \widetilde{s}, \pi, q) \xrightarrow{\epsilon,\epsilon}_{D'} ([s_1/x_1] \dots [s_n/x_n] s_0, \pi, q) \xrightarrow{\widetilde{\ell}, R}{}^{N-1}_{D'} (h_{\theta,m}\, \widetilde{u}, \pi', q').$$

Since the mark does not affect the induced run of the automaton, we have

$$([s_1/x_1] \dots [s_n/x_n] s_0^\sharp, \pi, q) \xrightarrow{\widetilde{\ell}, R}{}^{N-1}_{D'} (s'', \pi', q')$$

for some s'' (which is equivalent to $h_{\theta,m}\, \widetilde{u}$ except for marks). Comparing this reduction sequence with

$$([s_1/x_1] \dots [s_n/x_n] s_0^\sharp, \pi, q) \xrightarrow{\widetilde{\ell}, R'}{}^{N-1}_{D'^\sharp} (h^\sharp_{\theta,m}\, \widetilde{u'}, \pi', q')$$

given above, we conclude that $s'' = h^\sharp_{\theta,m}\, \widetilde{u'}$ and $R = R'$. Thus

$$\Gamma' \cup ([\Xi^\sharp] \uparrow \mathbf{max}(\Omega(R))) \vdash_{\mathcal{A}} t' : q' \Rrightarrow h^\sharp_{\theta,m}\, \widetilde{u'}.$$

Hence $h : (\theta, m, m) \in ([\Xi^\sharp] \uparrow \mathbf{max}(\Omega(R)))$ and thus $m = \mathbf{max}(\Omega(R))$ by definition of $[\Xi^\sharp]$. □

Lemma 44. *Assume that $\Xi \vdash_{\mathcal{A}} P \Rrightarrow (P', \Omega)$. Let $P = (\mathbf{main}, D)$ and $P' = (\mathbf{main}', D')$. For each choice sequence $\pi \in \{\mathtt{L}, \mathtt{R}\}^\omega$, the following conditions are equivalent.*

1. *There exists an infinite reduction sequence*

$$\pi \Vdash (\mathbf{main}', q_I) = (s_0, q_0) \xrightarrow{\ell_1, R_1}{}^{*}_{D'} (s_1, q_1) \xrightarrow{\ell_2, R_2}{}^{*}_{D'} \cdots$$

   such that $R_1 R_2 \cdots$ is an infinite sequence and $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \cdots))$ is even.

2. *There exists an infinite call-sequence*

$$\pi \Vdash \mathbf{main}' = g^0_{\theta_0, m_0}\, \tilde{u}_0 \overset{k_0}{\rightsquigarrow}_{D'} g^1_{\theta_1, m_1}\, \tilde{u}_1 \overset{k_1}{\rightsquigarrow}_{D'} g^2_{\theta_2, m_2}\, \tilde{u}_2 \overset{k_2}{\rightsquigarrow}_{D'} \cdots .$$

   such that $\mathbf{maxInf}(\Omega(\widetilde{g}))$ is odd.

Proof. ((1) ⇒ (2)) Assume an infinite reduction sequence

$$\pi \Vdash (\mathbf{main}', q_I) = (s_0, q_0) \xrightarrow{\ell_1, R_1}{}^{*}_{D'} (s_1, q_1) \xrightarrow{\ell_2, R_2}{}^{*}_{D'} \cdots .$$

By Corollary 1, we have a (unique) infinite call-sequence

$$\pi \Vdash \mathbf{main}' = g^0_{\theta_0, m_0}\, \tilde{u}_0 \overset{k_0}{\rightsquigarrow}_{D'} g^1_{\theta_1, m_1}\, \tilde{u}_1 \overset{k_1}{\rightsquigarrow}_{D'} g^2_{\theta_2, m_2}\, \tilde{u}_2 \overset{k_2}{\rightsquigarrow}_{D'} \cdots .$$

So the given reduction sequence can be rewritten as

$$\pi \Vdash (\mathbf{main}, q_I) = (g^0_{\theta_0, m_0}\, \tilde{u}_0, q'_0) \xrightarrow{\widetilde{\ell'_1}, R'_1}{}^{k_0}_{D'} (g^1_{\theta_1, m_1}\, \tilde{u}_1, q'_1) \xrightarrow{\widetilde{\ell'_2}, R'_2}{}^{k_1}_{D'} (g^2_{\theta_2, m_2}\, \tilde{u}_2, q'_2) \xrightarrow{\widetilde{\ell'_3}, R'_3}{}^{k_2}_{D'} \cdots .$$

Note that $R_1 R_2 \cdots = R'_1 R'_2 \ldots$. By Lemma 43, $m_i = \mathbf{max}(\Omega_{\mathcal{A}}(R'_i))$. Since $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \cdots))$ is even,

$$\begin{aligned}
\mathbf{maxInf}(\Omega(\widetilde{g})) &= \mathbf{maxInf}(m_1 + 1, m_2 + 1, \ldots) \\
&= \mathbf{maxInf}(\mathbf{max}(\Omega_{\mathcal{A}}(R'_1)) + 1, \mathbf{max}(\Omega(R'_2)) + 1, \ldots) \\
&= \mathbf{maxInf}(\mathbf{max}(\Omega_{\mathcal{A}}(R'_1)), \mathbf{max}(\Omega_{\mathcal{A}}(R'_2)), \ldots) + 1 \\
&= \mathbf{maxInf}(\Omega_{\mathcal{A}}(R'_1 R'_2 \ldots)) + 1 \\
&= \mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1 R_2 \ldots)) + 1
\end{aligned}$$

is odd.

((2) ⇒ (1)) Assume an infinite call-sequence

$$\pi \Vdash \mathbf{main}' = g^0_{\theta_0, m_0}\, \tilde{u}_0 \overset{k_0}{\rightsquigarrow}_{D'} g^1_{\theta_1, m_1}\, \tilde{u}_1 \overset{k_1}{\rightsquigarrow}_{D'} g^2_{\theta_2, m_2}\, \tilde{u}_2 \overset{k_2}{\rightsquigarrow}_{D'} \cdots .$$

This is an infinite reduction sequence

$$\pi \Vdash \mathbf{main}' = g^0_{\theta_0, m_0}\, \tilde{u}_0 \xrightarrow{\widetilde{\ell_1}}{}^{k_0}_{D'} g^1_{\theta_1, m_1}\, \tilde{u}_1 \xrightarrow{\widetilde{\ell_2}}{}^{k_1}_{D'} g^2_{\theta_2, m_2}\, \tilde{u}_2 \xrightarrow{\widetilde{\ell_3}}{}^{k_2}_{D'} \cdots .$$

By the assumption on the program, $\widetilde{\ell_1}\widetilde{\ell_2}\dots$ is an infinite sequence. Then by Corollary 5,

$$\pi \Vdash (\mathbf{main}', q_0) = (g^0_{\theta_0,m_0}\ \tilde{u}_0, q_0) \xrightarrow{\widetilde{\ell_1},R_1 k_0}_{D'} (g^1_{\theta_1,m_1}\ \tilde{u}_1, q_1) \xrightarrow{\widetilde{\ell_2},R_2 k_1}_{D'} (g^2_{\theta_2,m_2}\ \tilde{u}_2, q_2) \xrightarrow{\widetilde{\ell_3},R_3 k_2}_{D'} \cdots$$

for some $\{(q_i, R_i)\}_i$. By Lemma 43, $m_i = \mathbf{max}(\Omega_{\mathcal{A}}(R_i))$. Hence

$$\begin{aligned}\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1R_2\dots)) &= \mathbf{maxInf}(\mathbf{max}(\Omega_{\mathcal{A}}(R_1)), \mathbf{max}(\Omega_{\mathcal{A}}(R_2)), \dots)\\ &= \mathbf{maxInf}(m_1m_2\dots)\\ &= \mathbf{maxInf}(\Omega(\widetilde{g})) - 1\end{aligned}$$

Since $\mathbf{maxInf}(\Omega(\widetilde{g}))$ is odd, $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1R_2\dots))$ is even. The sequence $R_1R_2\dots$ is infinite since $\widetilde{\ell_1}\widetilde{\ell_2}\dots$ is. □

Proof of Theorem 5 Assume that $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P_0, \Omega)$. Then by Lemma 37, there exists P_1 such that $\Xi \vdash_{\mathcal{A}} P \Rrightarrow (P_1, \Omega)$ and $P_0 = [\Box/\circ]P_1$. Obviously $\models_{csa} (P_1, \Omega)$ if and only if $\models_{csa} (P_0, \Omega)$. Let $P_1 = (D_1, \mathbf{main}_1)$.

We prove the lemma by establishing the equivalence of the following propositions:

1. $\mathbf{InfTraces}(P) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$.
2. $\exists\widetilde{\ell} \in \mathbf{InfTraces}(P).\exists R$: run of $\mathcal{A}$ over $\widetilde{\ell}$. $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R))$ is *even*.
3. There exist π and an infinite reduction sequence

   $$\pi \Vdash (\mathbf{main}_1, q_I) = (s_0, q_0) \xrightarrow{\ell_1,R_1}{}^*_{D_1} (s_1, q_1) \xrightarrow{\ell_2,R_2}{}^*_{D_1} \cdots$$

   such that $R_1R_2\cdots$ is an infinite sequence and $\mathbf{maxInf}(\Omega_{\mathcal{A}}(R_1R_2\cdots))$ is even.
4. There exist π and an infinite call-sequence

   $$\pi \Vdash \mathbf{main}_1 = g^0_{\theta_0,m_0}\ \tilde{u}_0 \overset{k_0}{\rightsquigarrow}_{D_1} g^1_{\theta_1,m_1}\ \tilde{u}_1 \overset{k_1}{\rightsquigarrow}_{D_1} g^2_{\theta_2,m_2}\ \tilde{u}_2 \overset{k_2}{\rightsquigarrow}_{D_1} \cdots.$$

   such that $\mathbf{maxInf}(\Omega(\widetilde{g}))$ is odd.
5. $\neg(\models_{csa} (P_1, \Omega))$.
6. $\neg(\models_{csa} (P_0, \Omega))$.

(1) ⇔ (2): By definition.
(2) ⇔ (3): By Lemma 42.
(3) ⇔ (4): By Lemma 44.
(4) ⇔ (5): By definition
(5) ⇔ (6): Obvious.

E.5 Proof (Sketch) of Theorem 6

This follows by essentially the same argument as the proof of completeness of the intersection type system for trivial model checking of HORS ([19], Theorem 4.6); thus, we provide only a proof sketch.

First, we restrict intersection types by the following refinement relation.

Definition 26. *The* refinement relation $\theta :: \kappa$ *between intersection types and simple types is defined by:*

$$\frac{}{q :: \star}$$

$$\frac{\rho :: \eta \qquad \theta :: \kappa}{(\rho \to \theta) :: (\eta \to \kappa)}$$

$$\frac{}{\mathtt{int} :: \mathtt{int}}$$

$$\frac{\theta_i :: \kappa \textit{ for each } i \in \{1, \dots, k\}}{\bigwedge_{1 \le i \le k}(\theta_i, m_i) :: \kappa}$$

The relation is extended to the one on type environments by: $\Xi :: \mathcal{K}$ iff $\theta :: \mathcal{K}(x)$ for each $x : (\theta, m) \in \Xi$, and $\Gamma :: \mathcal{K}$ iff $\theta :: \mathcal{K}(x)$ for each $x : (\theta, m, m') \in \Gamma$.

Let $P = (D, t)$ be a program such that $\mathcal{K} \vdash P$. Here, we assume without loss of generality that t is a function symbol f_1. Let $\Xi_{max} = \{f : (\theta, m) \mid f : \kappa \in \mathcal{K}, \theta :: \kappa\}$, i.e., the largest type environment that refines $\mathcal{K}$. We define the function $\mathcal{F}$ on top type environments by:

$$\mathcal{F}(\Xi) = \{f : (\theta, m) \in \Xi \mid \Xi \vdash_{\mathcal{A}} D(f) : \theta \Rightarrow t'\}.$$

As in [19], $\mathcal{F}$ just filters out invalid type assumptions. To show Theorem 6, it is sufficient to prove the existence of a fixpoint Ξ of $\mathcal{F}$ such that $f_1 : q_I \in \Xi$ (which serves as a witness of Theorem 6).

To this end, we prepare a restriction of Theorem 6 for recursion-free programs:

Lemma 45. *Let $P = (D, f)$ be a recursion-free program such that $\mathcal{K} \vdash P$, and $\mathcal{A}$ be a non-deterministic parity automaton. Then, there exists Ξ, P' and Ω such that $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega)$.*

Proof (Sketch). Consider a non-standard reduction relation $t \longrightarrow_{ns,D} t'$, where conditionals, non-deterministic branches, and events are just considered tree constructors. Thus, the reduction rule just consists of:

$$\frac{f\,\widetilde{x} = s \in D \qquad |\widetilde{x}| = |\widetilde{t}|}{E[f\,\widetilde{t}] \longrightarrow_{ns,D} E[[\widetilde{t}/\widetilde{x}]s]} \qquad \text{(R-Fun)}$$

where E ranges over the set of evaluation contexts defined by:

$$\begin{aligned} E ::= {} & [\,] \mid E \Box t \mid t \Box E \mid \textbf{event } a; E \\ & \mid \textbf{if } p(t'_1, \dots, t'_k) \textbf{ then } E \textbf{ else } t \mid \textbf{if } p(t'_1, \dots, t'_k) \textbf{ then } t \textbf{ else } E. \end{aligned}$$

Since P is recursion-free and simply-typed, there must be a non-standard reduction sequence $f \longrightarrow^*_{ns,D} t \not\longrightarrow_{ns,D}$. Since the term t does not contain any function symbol, we have $\emptyset \vdash_{\mathcal{A}} (D, t) \Rightarrow (P'', \emptyset)$. By the subject expansion property (i.e., the preservation of the transformation relation by the inverse of reductions, which we omit to prove, as it is standard), we have $\Xi \vdash_{\mathcal{A}} P \Rightarrow (P', \Omega)$ for some Ξ such that $\Xi :: \mathcal{K}$, as required. □

Now, let $P^{(n)}$ be the recursion-free program defined in the proof of Theorem 1, where $n = |\Xi_{max}| + 1$. Let Ξ' be the witness type environment Ξ for $P^{(n)}$ given in Lemma 45. Define $\Xi_0, \ldots, \Xi_n$ by: $\Xi_k = \{f : (\theta, m) \mid f^{(j)} : (\theta, m) \in \Xi', j \geq k\}$. Then, we have (i) $\Xi_{k+1} \subseteq \mathcal{F}(\Xi_k) \subseteq \Xi_k$ for each $k < n$ (the first equality follows from the fact that the body of $f^{(k+1)}$ is typed by using only type bindings for $f_1^{(k)}, \ldots, f_\ell^{(k)}$; the second equality follows from the definition of $\mathcal{F}$), (ii) $f_1 : q_I \in \Xi_n$, and (iii) $\Xi_k \subseteq \Xi_{max}$, and (iv) $\Xi_n \subseteq \Xi_{n-1} \subseteq \cdots \subseteq \Xi_0$. From the conditions (iii), (iv) and $n = |\Xi_{max}| + 1$, there must exist k such that $\Xi_k = \Xi_{k+1}$. Thus, by this, (i), and (ii), Ξ_k is a fixpoint of $\mathcal{F}$ and contains $f_1 : q_I$. This completes the proof of Theorem 6.

The witness type environment $\Xi^{(n)}$ is too large in general. In practice, one can find a smaller witness type environment by using a practical HORS model checking algorithm [7, 19, 34]; by regarding conditionals, non-determinism, and events as tree constructors, we can see the problem of finding a witness type environment as that of finding a witness type environment for HORS model checking.

F An Example of the Translation of Section 7

Here we show derivation trees for the translation in Example 11.

The body of g is translated as follows (where we omit irrelevant type bindings), where $\Gamma_0 = k : (q_a, 0, 0), k : (q_b, 1, 0)$.

$$\dfrac{\dfrac{k : (q_a, 0, 0), k : (q_b, 1, 0) \vdash_{\mathcal{A}} k : q_a \Rightarrow k_{q_a,0}}{\Gamma_0 \vdash_{\mathcal{A}} (\textbf{event } \mathsf{a}; k) : q_a \Rightarrow (\textbf{event } \mathsf{a}; k_{q_a,0})} \quad \dfrac{k : (q_a, 0, 1), k : (q_b, 1, 1) \vdash_{\mathcal{A}} k : q_b \Rightarrow k_{q_b,1}}{\Gamma_0 \vdash_{\mathcal{A}} (\textbf{event } \mathsf{b}; k) : q_a \Rightarrow (\textbf{event } \mathsf{a}; k_{q_b,1})}}{\Gamma_0 \vdash_{\mathcal{A}} (\textbf{event } \mathsf{a}; k) \Box (\textbf{event } \mathsf{b}; k) : q_a \Rightarrow (\textbf{event } \mathsf{a}; k_{q_a,0}) \Box (\textbf{event } \mathsf{b}; k_{q_b,1})}$$

The body of f is translated as follows.

$$\dfrac{\Gamma_1, x : \mathtt{int} \vdash_{\mathcal{A}} x : \mathtt{int} \Rightarrow x_{\mathtt{int}} \quad \Gamma_1, x : \mathtt{int} \vdash_{\mathcal{A}} 0 : \mathtt{int} \Rightarrow 0 \quad \pi_1 \quad \pi_2}{\Gamma_1, x : \mathtt{int} \vdash_{\mathcal{A}} \textbf{if } x > 0 \textbf{ then } g\,(f(x-1)) \textbf{ else } (\textbf{event } \mathsf{b}; f\,5) : q_a \Rightarrow t_{f,q_a}}$$

Here, Γ_1 is:

$$g : ((q_a, 0) \wedge (q_b, 1) \rightarrow q_a, 0, 0), \quad g : ((q_a, 0) \wedge (q_b, 1) \rightarrow q_b, 0, 0),$$
$$f : (\mathtt{int} \rightarrow q_a, 0, 0), \quad f : (\mathtt{int} \rightarrow q_b, 1, 0)$$

and π_1 and π_2 are:

$$\pi_1 = \dfrac{\Gamma_1, x : \mathtt{int} \vdash_{\mathcal{A}} g : (q_a, 0) \wedge (q_b, 1) \rightarrow q_a \Rightarrow g_{(q_a,0)\wedge(q_b,1)\rightarrow q_a,0} \quad \pi_3 \quad \pi_4}{\Gamma_1, x : \mathtt{int} \vdash_{\mathcal{A}} g\,(f(x-1)) : q_a \Rightarrow g_{(q_a,0)\wedge(q_b,1)\rightarrow q_a,0}\,(f_{\mathtt{int}\rightarrow q_a,0}(x_{\mathtt{int}} - 1))\,(f_{\mathtt{int}\rightarrow q_b,1}(x_{\mathtt{int}} - 1))}$$

$$\pi_2 = \cfrac{\cfrac{\Gamma_1 \uparrow 1, x:\mathtt{int} \vdash_{\mathcal{A}} f:\mathtt{int} \to q_b \Rightarrow f_{\mathtt{int}\to q_b,1} \quad \Gamma_1 \uparrow 1, x:\mathtt{int} \vdash_{\mathcal{A}} 5:\mathtt{int} \Rightarrow 5}{\Gamma_1 \uparrow 1, x:\mathtt{int} \vdash_{\mathcal{A}} f\,5 : q_a \Rightarrow f_{\mathtt{int}\to q_b,1}5}}{\Gamma_1, x:\mathtt{int} \vdash_{\mathcal{A}} \textbf{event}\ \mathsf{b}; f\,5 : q_a \Rightarrow \textbf{event}\ \mathsf{b}; f_{\mathtt{int}\to q_b,1}5}$$

$$\pi_3 = \cfrac{\Gamma_1 \uparrow 0, x:\mathtt{int} \vdash_{\mathcal{A}} f:\mathtt{int} \to q_a \Rightarrow f_{\mathtt{int}\to q_a,0} \quad \cfrac{\cdots}{\Gamma_1 \uparrow 0, x:\mathtt{int} \vdash_{\mathcal{A}} x-1:\mathtt{int} \Rightarrow x_{\mathtt{int}}-1}}{\Gamma_1 \uparrow 0, x:\mathtt{int} \vdash_{\mathcal{A}} f(x-1) : q_a \Rightarrow f_{\mathtt{int}\to q_a,0}(x_{\mathtt{int}}-1)}$$

$$\pi_4 = \cfrac{\Gamma_1 \uparrow 1, x:\mathtt{int} \vdash_{\mathcal{A}} f:\mathtt{int} \to q_b \Rightarrow f_{\mathtt{int}\to q_b,1} \quad \cfrac{\cdots}{\Gamma_1 \uparrow 1, x:\mathtt{int} \vdash_{\mathcal{A}} x-1:\mathtt{int} \Rightarrow x_{\mathtt{int}}-1}}{\Gamma_1 \uparrow 1, x:\mathtt{int} \vdash_{\mathcal{A}} f(x-1) : q_b \Rightarrow f_{\mathtt{int}\to q_b,1}(x_{\mathtt{int}}-1)}$$

G Proving $\mathrm{HFL}_{\mathbf{Z}}$ formulas in Coq

Given program verification problems, the reductions presented in Sections 5–7 yield $\mathrm{HFL}_{\mathbf{Z}}$ model checking problems, which may be thought as a kind of "verification conditions" (like those for Hoare triples). Though we plan to develop automated/semi-automated tools for discharging the "verification conditions", we demonstrate here that it is also possible to use an interactive theorem prover to do so.

Here we use Coq proof assistant, and consider $\mathrm{HFL}_{\mathbf{Nat}}$(HFL extended with natural numbers) instead of $\mathrm{HFL}_{\mathbf{Z}}$. Let us consider the termination of the following program:

```
let sum n k =
  if n<=0 then k 0
  else sum (n-1) (fun r -> k(n+r))
in sum m (fun r->())
```

Here, we assume m ranges over the set natural numbers.

The translation in Section 5.2 yields the following $\mathrm{HFL}_{\mathbf{Nat}}$ formulas.

$$(\mu \mathtt{sum}.\lambda n.\lambda k.(n \le 0 \Rightarrow k\,0) \wedge (n > 0 \Rightarrow \mathtt{sum}\,(n-1)\,\lambda r.k(n+r)))\,m\,(\lambda r.\mathbf{true}).$$

The goal is to prove that for every m, the formula is satisfied by the trivial model $\mathrm{L}_0 = (\{\mathtt{s}_\star\}, \emptyset, \emptyset, \mathtt{s}_\star)$.

In order to avoid the clumsy issue of representing variable bindings in Coq, we represent the *semantics* of the above formula in Coq. The following definitions correspond to those of $\mathcal{D}_{\mathrm{L},\tau}$ in Section 2.

```
(* syntax of simple types:
   "arint t" and "ar t1 t2" represent nat->t and t1->t2 respectively *)
Inductive ty: Set :=
```

```
    o: ty
  | arint: ty -> ty
  | ar: ty -> ty -> ty.

(* definition of semantic domains, minus the monotonicity condition *)
Fixpoint dom (t:ty): Type :=
  match t with
     o => Prop
   | arint t' => nat -> dom t'
   | ar t1 t2 => (dom t1) -> (dom t2)
  end.
```

Here, we use Prop as the semantic domain $\mathcal{D}_{L_0,\bullet} = \{\emptyset, \{s_\star\}\}$ and represent $\{s_\star\}$ as a proposition True.

Above, we have deliberately omitted the monotonicity condition, which is separately defined by induction on simple types, as follows.

```
Fixpoint ord (t:ty) {struct t}: dom t -> dom t -> Prop :=
  match t with
    o => fun x: dom o => fun y: dom o => (x -> y)
  | arint t' =>
     fun x: dom (arint t') => fun y: dom (arint t') =>
          forall z:nat, ord t' (x z) (y z)
  | ar t1 t2 =>
      fun x: dom (ar t1 t2) => fun y: dom (ar t1 t2) =>
       forall z w:dom t1, ord t1 z z -> ord t1 w w ->
       ord t1 z w -> ord t2 (x z) (y w)
  end.

Definition mono (t: ty) (f:dom t) :=
    ord t f f.
```

Here, ord τ corresponds to $\sqsubseteq_{L_0,\tau}$ in Section 2,[13] and the monotonicity condition on f is expressed as the reflexivity condition ord $\tau\ f\ f$.

We can then state the claim that the sum program is terminating for every m as the following theorem:

```
Definition sumt := arint (ar (arint o) o).
Definition sumgen :=
  fun sum: dom sumt =>
   fun n:nat=> fun k:nat->Prop =>
    (n<=0 -> k 0) /\ (n>0 -> sum (n-1) (fun r:nat=>k(r+n))).

Theorem sum_is_terminating:
```

[13] Note, however, that since dom t may be inhabited by non-monotonic functions, "ord τ" is not reflexive.

```
  forall sum: dom sumt,
  forall FPsum: (* sum is a fixpoint of sumgen *)
     (forall n:nat, forall k:nat->Prop, sum n k <-> sumgen sum n k),
  forall LFPsum: (* sum is the least fixpoint of sumgen *)
     (forall x:dom sumt,
       mono sumt x ->
       ord sumt (sumgen x) x -> ord sumt sum x),
  forall m:nat, sum m (fun r:nat => True).
(* can be automatically generated up to this point *)
Proof.
(* this part should be filled by a user *)
...
Qed.
```

Here, `sumgen` is the semantics of the argument of the μ-operator (i.e., $\lambda \texttt{sum}.\lambda n.\lambda k.(n \le 0 \Rightarrow k\,0) \land (n > 0 \Rightarrow \texttt{sum}\,(n-1)\,\lambda r.k(n+r)))$, and the first three "`forall ...`" assumes that `sum` is the least fixpoint of it, and the last line says that `sum` m $\lambda x.$`True` is equivalent to `True` for every m.

Note that except the proof (the part "`...`"), all the above script can be *automatically* generated based on the development in the paper (like `Why3`[50], but *without* any invariant annotations).

The following is a proof of the above theorem.

```
Proof.
  intros.
  (* apply induction on m *)
  induction m;
  apply FPsum;
  unfold sumgen; simpl;auto.
  split; auto.
  omega.
  (* induction step *)
  assert (m-0=m); try omega.
  rewrite H; auto.
Qed.
```

More examples are found at `http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/hfl_in_coq.zip` (provided also as supplementary materials). The Coq proofs for some of those examples are much longer. For each example, however, there are only a few places where human insights are required (like "`induction m`" above). We, therefore, expect that the proofs can be significantly shortened by preparing appropriate libraries.

H HORS- vs HFL-based Approaches to Program Verification

In this section, we provide a more detailed comparison between our new HFL-based approach and HORS-based approaches [18, 19, 23, 27, 31, 33, 44] to program verification. Some familiarity with HORS-based approaches may be required to fully understand the comparison.

HORS model checking algorithms [22, 32] usually consist of two phases, one for computing a kind of higher-order "procedure summaries" in the form of variable profiles [32] or intersection types [22] (that summarize which states are visited between two function calls), and the other for solving games, which consists in nested least/greatest fixpoint computations. In the case of finite-data programs, the combination of the two phases provides a sound and complete verification algorithm [22, 32]. To deal with infinite-data programs, however, the HORS-based approaches [18, 19, 23, 27, 31, 33, 44] had to apply various transformations to map verification problems to HORS model checking problems in a sound but incomplete manner (incompleteness is inevitable because, in the presence of values from infinite data domains, the former is undecidable whereas the latter is decidable), as illustrated on the lefthand side of Figure 9. A problem about this approach is that the second phase of HORS model checking – nested least/greatest fixpoint computations – actually does not help much to solve the original problem, because least fixpoint computations are required for proving liveness (such as termination), but the liveness of infinite-data programs usually depends on properties about infinite data domains such as "there is no infinite decreasing sequence of natural numbers," which are not available after the transformations to HORS. For this reason, the previous techniques for proving termination [27] and fair termination [31] used HORS model checking only as a backend of a safety property checker, where only greatest fixpoint computations are performed in the second phase; reasoning about liveness was performed during the transformation to HORS model checking.

As shown on the righthand side of Figure 9, in our new HFL-based approach, we have extended the first phase of HORS model checking – the computation of higher-order procedure summaries – to deal with infinite data programs, and moved it up front; we then formalized the remaining problems as HFL model checking problems. That can be viewed as the essence of the reduction in Section 7, and the reductions in Sections 5 and 6 are those for degenerate cases. Advantages of the new approach include: (i) necessary information on infinite data is available in the second phase of least/greatest computations (cf. the discussion above on the HORS-based approach), and (ii) various verification problems boil down to the issue of how to prove least/greatest fixpoint formulas; thus we can reuse and share the techniques developed for different verification problems. The price to pay is that the first phase (especially a proof of its correctness) is technically more involved, because now we need to deal with infinite-data programs instead of HORS's (which are essentially finite-data functional programs). That explains long proofs in Appendices (especially Appendix D and E).

Additional References for Appendices

46. Jurdzinski, M.: Small progress measures for solving parity games. In: Proceeding of STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer (2000)

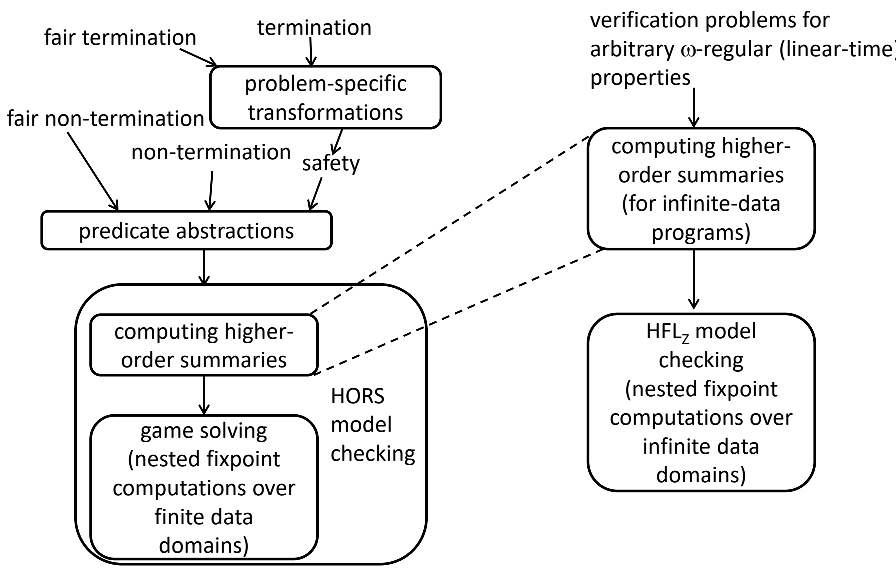


Fig. 9. Comparison between HORS- and HFL-based approaches. In the HORS-based approaches (shown on the lefthand side), there are actually some feedback loops (due to counterexample-guided abstraction refinement, etc.), which are omitted. The right hand side shows the approach based on the reduction in Section 7. In the reductions in Section 5 and 6, the first phase is optimized for degenerate cases.

47. Kobayashi, N., Ong, C.H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. http://www-kb.is.s.u-tokyo.ac.jp/~koba/tmp/lics09-full.pdf. A longer version of [22] (2012)
48. Salvati, S., Walukiewicz, I.: Krivine machines and higher-order schemes. Information and Computation 239(Supplement C), 340 – 355 (2014), http://www.sciencedirect.com/science/article/pii/S0890540114000984
49. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2012)
50. Filliâtre, J.C., Paskevich, A.: Why3 — where programs meet provers. In: Felleisen, M., Gardner, P. (eds.) Proceedings of ESOP 2013. LNCS, vol. 7792, pp. 125–128. Springer (2013)
51. Winskel, G.: The Formal Semantics of Programming Languages: An Introduction. The MIT Press (1993)
52. Winskel, G.: Prime algebraicity. Theor. Comput. Sci. 410(41), 4160–4168 (2009)