Fully Abstract Normal Form Bisimulation for Call-by-Value PCF

Vasileios Koutavas
Trinity College Dublin
Dublin, Ireland
Email: Vasileios.Koutavas@tcd.ie

Yu-Yang Lin
Trinity College Dublin
Dublin, Ireland
Email: linhouy@tcd.ie

Nikos Tzevelekos
Queen Mary University of London
London, UK
Email: Nikos.Tzevelekos@qmul.ac.uk

***Abstract*—We present the first fully abstract normal form bisimulation for call-by-value PCF (PCF$_\mathsf{v}$). Our model is based on a labelled transition system (LTS) that combines elements from applicative bisimulation, environmental bisimulation and game semantics. In order to obtain completeness while avoiding the use of semantic quotiening, the LTS constructs traces corresponding to interactions with possible functional contexts. The model gives rise to a sound and complete technique for checking of PCF$_\mathsf{v}$ program equivalence, which we implement in a bounded bisimulation checking tool. We test our tool on known equivalences from the literature and new examples.**

I. Introduction

The full abstraction problem for PCF, i.e. constructing a denotational model that captures contextual equivalence in the paradigmatic functional language PCF, was put forward by Milner in the mid 1970's [28]. The first fully abstract denotational models for PCF were presented in the early 1990's and gave rise to the theory of *game semantics* [2], [12], [30], while fully abstract models for its call-by-value variant were given in [10], [3]. Fully abstract operational models of PCF have been given in terms of *applicative bisimulations* [1], [9], [11] and *logical relations* [31], and for other pure languages in terms of *environmental bisimulations* [39], [36] and *logical relations* [32], [4]. On the other hand, Loader demonstrated that contextual equivalence for finitary PCF is undecidable [26].

A limitation of the game semantics models for PCF is their intentional nature. While the denotations of inequivalent program terms are always distinct, there are equivalent terms whose denotations are also distinct and become equivalent only after a semantic quotiening operation. Quotiening requires universal quantification over tests, which amounts to quantification over all (innocent) contexts. This hinders the use of game models for pure functional languages to prove equivalence of terms, as any reasoning technique needs to involve all contexts of term denotations in the semantic model (i.e. all possible *Opponent strategies*). In a more recent work, Churchill et al. [7] were able to give a direct characterisation of program equivalence in terms of so-called *sets of O-views*, built out of term denotations. The latter work is to our knowledge the only direct (i.e. quotient-free) semantic characterisation of PCF contextual equivalence, though it is arguably more of theoretical value and does not readily yield a proof method.

This publication has emanated from research supported in part by a grant from Science Foundation Ireland under Grant number 13/RC/2094_2.

Operational models also involve quantification over all identical (applicative bisimulation) or related (logical relations, environmental bisimulations) closed arguments of type A, when describing the equivalence class of type $A \to B$. Although successful proof techniques of equivalence have been developed based on these models, universal quantification over opponent-generated terms must be handled in proofs with rather manual inductive or coinductive arguments.

Normal-Form (NF) bisimulation, also known as open bisimulation, was originally defined for characterising Lévy-Longo tree equivalence for the lazy lambda calculus [35] and adapted to languages with call-by-name [21], call-by-value [22], nondeterminism [23], aspects [14], recursive types [25], polymorphism [20], control with state [37], state-only [6], and control-only [5]. It has also been used to create equivalence verification techniques for a lambda calculus with state [16].

The main advantage of NF bisimulation is that it does away with quantification over opponent-generated terms, replacing them with fresh open names. This has also been shown [25], [20], [16] to relate to operational game semantics models where opponent-generated terms are also represented by names [19], [8], [13]. The main disadvantage of NF bisimulation is that — with the notable exception of languages with control and state [37], and state-only [6], [16] — it is too discriminating thus failing to be fully abstract with respect to contextual equivalence. This is particularly true for pure languages such as PCF, and its call-by-value variant PCF$_\mathsf{v}$ which is the target of this paper.

However, the discriminating power of NF bisimulation depends on the labelled transition system (LTS) upon which it is defined. Existing work define NF bisimulation over LTSs that treat call and return moves between term and context in a fairly standard way: these are immediately observable by the bisimulation as they appear in transition annotations, and context moves correspond to imperative, not purely functional, contexts. As we show in Section II, this is overly discriminating for a language such as PCF$_\mathsf{v}$. Moreover, existing NF bisimulation techniques, either do not make extensive use of the the context's knowledge in the LTS configurations (e.g. [25]), or consider an ever-increasing context knowledge (e.g. [6]) which is only fully abstract for imperative contexts.

In this paper we present the first fully abstract NF bisimulation for PCF$_\mathsf{v}$, defined in Section III. To achieve this we develop in Section IV a novel Labelled Transition System (LTS) which:

- is based on an operational presentation of game models (cf. [19]) and uses Proponent and Opponent configurations (and *call/return moves*) for evaluation steps that depend on the modelled term and its environment, respectively;
- uses an explicit stack principle to guarantee well-bracketing and stipulates that the *opponent view* of the LTS trace be restricted to moves related to the last open proponent call (cf. *well-bracketing* and *visibility* [12]);
- restricts opponent moves so that they correspond to those of a pure functional context, by explicitly keeping track of previous opponent responses to proponent moves and their corresponding (opponent) view (cf. *innocence* [12]);
- postpones observations of proponent call/return moves until computations are complete to avoid unnecessary distinctions between equivalent terms.

We then define a notion of NF bisimulation over this LTS which combines standard move/label synchronisation with coherence of corresponding opponent behaviours. We show that the latter is fully abstract with respect to contextual equivalence (Section V). Due to its operational nature and the absence of quantification over opponent-generated terms, the model lends itself to a bounded model-checking technique for equivalence which we implement in a prototype tool (Section VI). We conclude in Section VII.

II. Motivating Examples

We start with a simple example of equivalence that showcases unobservable behaviour differences in PCF$_\mathsf{v}$.

Example 1. Consider the following equivalent terms of type $(\mathsf{unit} \to \mathsf{int}) \to (\mathsf{unit} \to \mathsf{int}) \to \mathsf{int}$.

$M_1 =$
```
fun f -> fun g -> if f () == g () then
                  if f () == g () then 0 else 1
                  else 2
```

$N_1 =$
```
fun f -> fun g -> if g () == f () then 0 else 2
```

These two terms are contextually equivalent because the context cannot observe whether `f` and `g` have been called more than once with the same argument. Two calls of a pure and deterministic function with the same argument both diverge or return the same value. Moreover, the context cannot observe the order of calls to the context-generated functions `f` and `g`. As we will see in Section IV, our LTS restricts the behaviour of context-generated functions such as `f` and `g` so that they behave in a pure deterministic manner, and does not make distinctions based on their call order. □

We now discuss key *observable* behaviour differences in PCF$_\mathsf{v}$ through the lens of bisimulation theories. As explained in [15], the main feature in environmental bisimulation definitions [38], [39], [18], [17], [36] is *knowledge accumulation:* environmental bisimulation collects the term-generated functions in an environment representing the knowledge of the context. This knowledge is used in the following bisimulation tests to distinguish terms:

1) to call a function from the environment multiple times in a row with the same argument;
2) to call a function from the environment multiple times in a row with different arguments;
3) to call environment functions after other environment functions have returned; and
4) to use environment functions in the construction of context-generated functions.

The above is easily understood to be necessary in stateful languages and was shown to be needed in pure languages with existential types [15]. However, as applicative bisimulation has shown, it is unnecessary to accumulate the context's knowledge in order to create a theory of PCF$_\mathsf{v}$: applicative bisimulation interrogates related functions in isolation from other knowledge by simply applying them to identical arguments.

As discussed in the first example of this section, purity and determinism indeed make (1) unnecessary in PCF$_\mathsf{v}$. However, (2–4) are *necessary* tests that a normal form bisimulation theory for PCF$_\mathsf{v}$ must perform. This is because a normal form bisimulation definition must prescribe the necessary interaction between terms and context *under any evaluation context* and not just at top-level computations. Applicative bisimulation on the other hand is only defined in terms of top-level function applications, where the context's knowledge is limited. Universal quantification over the code of context-generated function arguments implicitly encodes all the interactions that related terms may have with these arguments. We showcase the need for (2–4) in the following three example inequivalences.

Example 2. Consider the inequivalent terms M_2, N_2 of type $(((\mathsf{bool} \to \mathsf{bool}) * (\mathsf{bool} \to \mathsf{bool})) \to \mathsf{bool}) \to \mathsf{bool} \to \mathsf{bool}$.

$M_2 =$
```
fun f -> fun b ->
  let rec X d = f (X, fun _ ->  d)
  in X b
```

$N_2 =$
```
fun f -> fun b ->
  f ((fun _ -> _bot_), fun _ ->  b)
```

Here `_bot_` is a diverging term and `_` represents an unused variable; `X` has type $\mathsf{bool} \to \mathsf{bool}$.

Term M_2 will receive a function `f` and a boolean `b`. It will then create a recursive term which calls `f` with a pair containing two $\mathsf{bool} \to \mathsf{bool}$ functions. If `f` calls `X`, the first function in the pair, with a boolean `d`, computation will recur; if it calls the second function, it will receive the argument of the latest call to `X`. On the other hand, N_2 calls `f` with a pair of functions where the first one diverges upon call, and the second one returns `b`, provided at the beginning of the interaction.

These terms can be distinguished by the following context:

```
let f = fun (X, fd) -> if fd false then X false
                       else true
in [] f true
```

This context creates a function `f` that receives two functions `X` and `fd`, and conditionally calls `X` with `false`, if the call to `fd` returns `true`. When placed in the hole `[]` of this context, M_2 will receive `f` and value `true`. Recursive function `X` will thus be first called with `true`, in the last line of M_2, and then again with `false` by `f`, causing the termination of the computation. On the other hand, with N_2 in the hole, the context will diverge.

This is effectively the only simple context that can distinguish M_2 and N_2, and thus a NF bisimulation theory of equivalence for PCF$_\mathsf{v}$ must accumulate `X` in the opponent's knowledge at inner interaction levels to allow calling `X` after `fd` has returned. This shows the need for allowing (3) in a NF bisimulation. Indeed, if we omit this from the technique we develop in the following sections, M_2 and N_2 would be deemed equivalent. □

The following variation of the above example shows that the context may need to call the same function twice, with different arguments, to make observations.

Example 3. Consider the inequivalent terms M_3, N_3 of type $((\mathsf{bool} \to (\mathsf{bool} \to \mathsf{bool})) \to \mathsf{bool}) \to \mathsf{bool} \to \mathsf{bool}$.

$M_3 =$
```
fun f -> fun b ->
  let rec X d = f (fun e -> if e then X
                             else (fun _ -> d))
  in X b
```

$N_3 =$
```
fun f ->  fun b ->
  f (fun e -> if e then (fun d -> _bot_)
             else (fun _ -> b))
```

where `X` has type $\mathsf{bool} \to \mathsf{bool}$. The distinguishing context is

```
let f = fun fXd ->
          let X = fXd true in
          let fd = fXd false in
          if fd false then X false else true
in [] f true
```

Here the interaction between the terms and the context are as in the previous example, with the difference that the context must apply `fXd` to `true` and then `false` to receive the two functions `X` and `fd`. The context terminates with M_3 but diverges with N_3 in its hole.

This is effectively the only simple context that can distinguish M_3 and N_3, and thus a NF bisimulation theory of equivalence for PCF$_\mathsf{v}$ that describes all the term-context interactions must accumulate `fXd` in the context's knowledge in order to apply it twice in a row. This showcases the need for allowing (2) in a NF bisimulation. □

Our final example shows that functions from the context's knowledge must be used within a context-generated function in order to distinguish two terms.

Example 4. Consider the inequivalent terms M_4, N_4 of type $T = ((\mathsf{int} \to \mathsf{int}) \to \mathsf{int} \to \mathsf{int}) \to \mathsf{int} \to \mathsf{int}$.

$M_4 =$
```
let rec X count = fun f -> fun i ->
    f  (fun j -> if (count > 0)
                 then X (count-1) f j
                 else _bot_)  i
in X k
```

$N_4 =$
```
fun f -> fun i -> let rec Y j = f Y j
                  in Y i
```

where `X` and `Y` have type $\mathsf{int} \to T$ and $\mathsf{int} \to \mathsf{int}$, respectively.

This is a family of examples in which the distinguishing interaction increases with k; N_4 enables `f` to call itself an arbitrary number of times, whereas M_4 enables up to k recursive calls of `f` before it diverges. The distinguishing context below attempts to perform $k+1$ recursive calls and then to return 0:

```
[] (fun Z -> fun i -> if i > 0 then Z (i-1) else 0)
   (k+1)
```

This context diverges with M_4 but converges with N_4 in its hole. To achieve this, the context uses the function received as argument `Z` inside the context-generated function `fun i -> if i > 0 then Z (i-1) else 0` which is given back to the term. As this is effectively the only context that can distinguish M_4 and N_4, we need to allow our NF bisimulation for PCF$_\mathsf{v}$ to construct (symbolic) function values that can internally refer to functions in the context's knowledge at the time of construction; showing the need for allowing (4) in a NF bisimulation. If we omit this from our technique, it would deem M_3 and N_3 equivalent. □

III. Language and Semantics

We work with the language PCF$_\mathsf{v}$, a simply-typed call-by-value lambda calculus with boolean and integer operations [10]. The syntax and reduction semantics are shown in Fig. 1. Expressions (Exp) include the standard lambda expressions with recursive functions ($\mathsf{fix} f(x).e$), together with standard base type constants (c) and operations ($op(\vec{e})$), as well as conditionals and tuple-deconstructing let expressions ($\mathsf{let}(\vec{x}) = e \;\mathsf{in}\; e$). We use standard macros, for example $\bot_T \stackrel{\text{def}}{=} \mathsf{fix} f_{\mathsf{unit} \to T}(x).fx$ and $\lambda x_T.e \stackrel{\text{def}}{=} \mathsf{fix} f_{T \to T'}(x).e$ (with f fresh for e).

The language PCF$_\mathsf{v}$ is simply-typed with typing judgements of the form $\Delta \vdash e : T$, where Δ is a type environment (omitted when empty) and T a value type (Type). The rules of the typing system are standard and omitted here [10]. Values consist of boolean, integer, and unit constants, functions and arbitrary length tuples of values.

The reduction semantics is by small-step transitions between closed expressions, $e \to e'$, defined using single-hole evaluation contexts (ECxt) over a base relation $\hookrightarrow$. Holes $[\cdot]_T$ are annotated with the type T of closed values they accept, which we omit when possible to lighten notation. Beta substitution of x with v in e is written as $e[v/x]$. We write $e \Downarrow$ to denote $e \to^* v$ for some v. We write $\vec{\chi}$ to mean a finite sequence of syntax objects $\chi_1, \ldots$, and assume standard syntactic sugar from the lambda calculus. In our examples we assume an ML-like syntax and implementation of the type system, which is also the concrete syntax of our prototype tool (Section VI).

$$
\begin{array}{rrcl}
\mathsf{Type}: & T & ::= & \mathsf{bool} \mid \mathsf{int} \mid \mathsf{unit} \mid T \to T \mid T_1 * \ldots * T_n \\
\mathsf{Exp}: & e, M, N & ::= & v \mid x \mid (\vec{e}) \mid op(\vec{e}) \mid e\,e \mid \mathsf{if}\, e\, \mathsf{then}\, e\, \mathsf{else}\, e \mid \mathsf{let}(\vec{x}) = e \text{ in } e \\
\mathsf{Val}: & u, v & ::= & c \mid \mathsf{fix} f_T(x).e \mid (\vec{v}) \\
\mathsf{ECxt}: & E & ::= & [\cdot]_T \mid (\vec{v}, E, \vec{e}) \mid op(\vec{v}, E, \vec{e}) \mid E\, e \mid v\, E \mid \mathsf{if}\, E\, \mathsf{then}\, e\, \mathsf{else}\, e \mid \mathsf{let}(\vec{x}) = E \text{ in } e \\
\mathsf{Cxt}: & D & ::= & [\cdot]_{i,T} \mid e \mid (\vec{D}) \mid op(\vec{D}) \mid D\, D \mid \mathsf{if}\, D\, \mathsf{then}\, D\, \mathsf{else}\, D \mid \mathsf{fix} f_T(x).D \mid \mathsf{let}(\vec{x}) = D \text{ in } D
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcll}
(\mathsf{fix} f(x).e)\, v & \hookrightarrow & e[v/x][\mathsf{fix} f(x).e/f] & op(\vec{c}) & \hookrightarrow & w & \text{if } op^{\mathsf{arith}}(\vec{c}) = w \\
\mathsf{let}(\vec{x}) = (\vec{v}) \text{ in } e & \hookrightarrow & e[\vec{v}/\vec{x}] & \mathsf{if}\, c\, \mathsf{then}\, e_1\, \mathsf{else}\, e_2 & \hookrightarrow & e_i & \text{if } (c, i) \in \{(\mathsf{tt}, 1), (\mathsf{ff}, 2)\} \\
 & & & E[e] & \to & E[e'] & \text{if } e \hookrightarrow e'
\end{array}
$$

Fig. 1. Syntax and reductions of $\mathsf{PCF_v}$. Variables x, y, z, etc. sourced from countably infinite set Var. c ranges over constants $()$, tt, ff and n (for any $n \in \mathbb{Z}$).

Contexts D contain multiple, non-uniquely indexed holes $[\cdot]_{i,T}$, where T is the type of value that can replace the hole (and each index i can have one related type). A context is called *canonical* if its holes are indexed $1, \ldots, n$, for some n. Given a canonical context D and a sequence of typed expressions $\Sigma \vdash \vec{e} : \vec{T}$, notation $D[\vec{e}]$ denotes the context D with each hole $[\cdot]_{i,T_i}$ replaced with e_i. We omit hole types where possible and indices when all holes in D are annotated with the same i. Standard contextual equivalence [29] follows.

Definition 5 (Contextual Equivalence)**.** Expressions $\vdash e_1 : T$ and $\vdash e_2 : T$ are *contextually equivalent*, written as $e_1 \equiv e_2 : T$, when for all contexts D such that $\vdash D[e_1] : \mathsf{unit}$ and $\vdash D[e_2] : \mathsf{unit}$ we have $D[e_1] \Downarrow$ iff $D[e_2] \Downarrow$. $\square$

Due to the language being purely functional, we can refine the contexts needed for contextual equivalence to *applicative* ones.

Definition 6. Applicative contexts are given by the syntax:

$$E_a ::= \ [\cdot]_T \mid E_a v \mid \mathsf{if}\ E_a = c\ \mathsf{then}\ ()\ \mathsf{else}\ \bot_{\mathsf{unit}} \mid \pi_i(E_a)$$

where $\pi_i(\chi)$ returns the i-th component of tuple χ. $\square$

Using the fact that applicative bisimulation is fully abstract [15], [11], we can show the following.

Proposition 7 (Applicative contexts suffice)**.** *$e_1 \equiv e_2 : T$ iff for all applicative contexts E_a such that $\vdash E_a[e_1]_T, E_a[e_2]_T : \mathsf{unit}$ we have $E_a[e_1] \Downarrow$ iff $E_a[e_2] \Downarrow$.*

IV. LTS WITH SYMBOLIC HIGHER-ORDER TRANSITIONS

We now define a Labelled Transition System (LTS) which allows us to probe higher-order values with possible symbolic arguments. The LTS follows the operational game semantics approach, with several adjustments:

- the basis of the LTS is the operational game model of [19];
- the Opponent behaviours are constrained to *innocent* ones (cf. [10]) by use of an *opponent memory* component M;
- the denotation of an expression is not just the transitions that the LTS produces for this expression but, instead, these transitions together with the corresponding opponent memory at top-level configurations.

Thus, the LTS comprises of *Proponent* and *Opponent* configurations with corresponding transitions, modelling the computations triggered by an expression and its context respectively. Opponent is construed as the syntactic context, which provides values for the functions that are open in the expression. Open functions are modelled with (opponent-generated) *abstract names*, which are accommodated by extending the syntax and typing rules with abstract function names α:

$$\mathsf{Val}: \quad u, v, w ::= c \mid \mathsf{fix} f_T(x).e \mid (\vec{v}) \mid \alpha^i_T$$

Abstract function names α^i_T are annotated with the type T of function they represent, and with an index $i \geq 0$ that is used for bookkeeping; these are omitted where not important. $\mathsf{an}(\chi)$ is the set of abstract names in χ.

The definition of our LTS (Fig. 2) is explained below.

Moves:

Our LTS uses *moves*:

$$\eta ::= \mathsf{call}(\alpha_T, D) \mid \mathsf{ret}(D) \mid \underline{\mathsf{call}}(i, v) \mid \underline{\mathsf{ret}}(v)$$

with contexts D and values v built from the following restricted grammars:

$$
\begin{aligned}
D_\bullet &::= c \mid [\cdot]_{i,T} \mid (\vec{D}_\bullet) \\
v_\bullet &::= c \mid \alpha_T \mid (\vec{v}_\bullet)
\end{aligned}
$$

Thus, $D_\bullet$ and $v_\bullet$ are values where functions are replaced by holes and abstract names, respectively. To lighten notation, we denote them by D, v.

Moves η are proponent call ($\mathsf{call}(\alpha, D)$) and return ($\mathsf{ret}(D)$) moves involved in transitions from opponent to proponent configurations; and opponent call ($\underline{\mathsf{call}}(i, v)$) and return ($\underline{\mathsf{ret}}(v)$) moves in transitions from opponent to proponent configurations.

Remark 8. Note the abstract names used in moves (and, later, traces) are of the form α_T, i.e. without i-annotations. This amounts to the fact that any two abstract names α^i_T, α^j_T with $i \neq j$ correspond to the same function played by opponent in two different points in the interaction. At each point, the proponent functions $\vec{v}$ that the opponent has access to may differ, and hence the need for different indices to distinguish the two instances of α_T. In the LTS, such distinction is not needed for proponent higher-order values as they are suppressed from proponent moves altogether. $\square$

Definition 9 (Traces)**.** We let a *trace* t be an alternating sequence of opponent/proponent moves. We write $t + t'$ or, sometimes for brevity, $t\,t'$ to mean trace concatenation. $\square$

Configurations:

Proponent configurations are written as $\langle A\,;M\,;K\,;t\,;e\,;V\rangle$ and proponent configurations as $\langle A\,;M\,;K\,;t\,;V\,;\vec{u}\rangle$. All configurations are ranged over by C. In these configurations:

- A is a partial map which assigns a sequence of names $\vec{v}$ to each abstract function name α (that has been used so far in the interaction) and integer index j. We write $\alpha^{j,\vec{v}} \in A$ for $A(\alpha, j) = \vec{v}$. The index j is used to distinguish between different uses of the same abstract function name α by opponent in the interaction. The sequence of values $\vec{v}$ represents the proponent functions that were available to opponent when the name α^j was used (knowledge accumulation for constructing context-generated functions, cf. Example 4). We write $A \uplus \alpha^{j,\vec{v}}$ for $A \cup ((\alpha, j), \vec{v})$, assuming $(\alpha, j) \notin A$.
- t is the *opponent-visible trace*, i.e. a subset of the current interaction that the opponent can have access to, starting with a move where the proponent calls an opponent abstract function.
- K is a stack of proponent continuations, created by nested proponent calls. We call configurations with an empty stack *top-level* and those with a non-empty stack *inner-level*; opponent top-level configurations are also called *final*. Configurations of the form $\langle\cdot\,;\cdot\,;\cdot\,;\cdot\,;e\,;\cdot\rangle$ are called *initial*.
- M is a set of opponent-visible traces. It ensures pure behaviour of the opponent (cf. Example 1): it restricts the moves of the opponent when an opponent-visible trace is being run for a second (or subsequent) time. Component M is also examined by the bisimulation to determine equivalence of top-level configurations. It can be seen as a *memory* of the behaviour of the opponent abstract functions so far and an oracle for future moves. Given M, we define a map from proponent-ending traces to next opponent moves:

$$\mathsf{next}_M(t) = \{\eta \mid t\eta \in M\}$$

  We consider only *legal* M's whereby $|\mathsf{next}_M(t)| \leq 1$ for any trace t ending in a proponent move and each abstract function name α appears at most once in M. We write $M[t]$ for $M \cup \{t\}$. We may also write M_C for the M-component of a configuration C.
- e is the proponent expression reduced in proponent configurations.
- In opponent configurations, $\vec{u}$ is the sequence of values (proponent functions) that are available to opponent to call at the given point in the interaction. In both kinds of configurations, V is a stack of sequences of proponent functions. These components encode the opponent knowledge accumulation necessary for a sound NF bisimulation theory for PCF$_\mathsf{v}$. They enable sequence of calls to proponent functions (cf. Examples 2 and 3), and construction of opponent-generated abstract functions with the appropriate level of knowledge attached to them (cf. Example 4).

Transitions:

Transitions are of the form $C \xrightarrow{l} C'$, where transition label l is either an immediately observable move η or a generic τ, hiding any move involved in the transition. In the former case, observable moves can be opponent calls ($\underline{\mathsf{call}}$) or proponent returns ($\mathsf{ret}$). Unlike standard LTSs, this LTS hides call/return moves involved in transitions of inner-level configurations, which are stored in the configuration memory M instead. As we will see later in this section, this is to allow equivalent terms to have different order of calls to opponent functions. Only *top-level* transitions contain move annotations, making them directly observable. These are transitions produced by one of the barbed rules (PropRetBarb, OpCallBarb). In the remaining transition rules moves are accumulated in traces which are stored in the memory component M of the configurations. These will be examined by the bisimulation at top-level configurations.

The simplest transitions are those produced by the PropTau rule, embedding reductions into proponent configurations. The remaining transitions involve interactions between opponent and proponent and are detailed below.

Proponent Return:

When the proponent expression has been reduced to a value, the LTS performs a ret-move, either by the PropRetBarb transition, when the configuration is top-level, or the PropRet transition, when it is not. In both cases the value v being returned is deconstructed to:

- an *ultimate pattern* D (cf. [24]), which is a context obtained from v by replacing each function in v with a distinct numbered hole; together with
- a sequence of values $\vec{v}$ such that $D[\vec{v}] = v$.

We let $\mathsf{ulpatt}(v)$ be a deterministic function performing this decomposition.

In rule PropRetBarb the functions $\vec{v}$ obtained from v become the knowledge of the resulting opponent configuration; opponent can call one of these functions to continue the interaction. The previous knowledge $\vec{u}$ stored in the one-frame stack is being dropped. This dropping of knowledge is sufficient for a sound NF bisimulation theory based on this LTS, as justified by our soundness result and corroborated by the conditions of applicative bisimulation which encode top-level interactions without accumulating opponent knowledge from previous moves.

On the other hand, in PropRet, $\vec{v}$ is added to the most current opponent knowledge $\vec{u}$, stored in the top-frame of the knowledge stack which is popped in the resulting configuration. This is necessary because, in inner level configurations, opponent should be allowed to call a proponent function it knew before it called the function that returned v, allowing observations such as those in Examples 2 and 3.

In PropRetBarb the context D extracted by ultimate pattern matching becomes observable in the transition label $\mathsf{ret}(D)$. Again, this is in line with the definition of applicative bisimulation where the return values of top-level functions are

$$\frac{e \to e'}{\langle A\,;M\,;K\,;t\,;e\,;V\rangle \xrightarrow{\tau} \langle A\,;M\,;K\,;t\,;e'\,;V\rangle}\ \textsc{PropTau}$$

$$\frac{(D,\vec{v}) = \mathsf{ulpatt}(v)}{\langle A\,;M\,;\cdot\,;t\,;v\,;\vec{u}\rangle \xrightarrow{\mathsf{ret}(D)} \langle A\,;M\,;\cdot\,;\cdot\,;\cdot\,;\vec{v}\rangle}\ \textsc{PropRetBarb}$$

$$\frac{(D,\vec{v}) = \mathsf{ulpatt}(v) \qquad K \neq \cdot \qquad t' = t + \mathsf{ret}(D)}{\langle A\,;M\,;K\,;t\,;v\,;\vec{u},V\rangle \xrightarrow{\tau} \langle A\,;M[t']\,;K\,;t'\,;V\,;\vec{u},\vec{v}\rangle}\ \textsc{PropRet}$$

$$\frac{(D,\vec{v}) = \mathsf{ulpatt}(v) \qquad t' = \mathsf{call}(\alpha, D) \qquad \vec{\alpha}^{j,\vec{u}} \in A}{\langle A\,;M\,;K\,;t\,;E[\alpha^j_{T_1\to T_2}\,v]\,;V\rangle \xrightarrow{\tau} \langle A\,;M[t']\,;(t,E[\cdot]_{T_2}),K\,;t'\,;V\,;\vec{u},\vec{v}\rangle}\ \textsc{PropCall}$$

$$\frac{\mathsf{next}_M(t) \subseteq \{\underline{\mathsf{ret}}(D[\vec{\alpha}])\} \qquad (D,\vec{\alpha}) \in \mathsf{ulpatt}(T') \qquad t'' = t + \underline{\mathsf{ret}}(D[\vec{\alpha}])}{\langle A\,;M\,;(t',E[\cdot]_{T'},T),K\,;t\,;V\,;\vec{v}\rangle \xrightarrow{\tau} \langle A \uplus \vec{\alpha}^{j,\vec{v}}\,;M[t'']\,;K\,;t'\,;E[D[\vec{\alpha}^j]]\,;V\rangle}\ \textsc{OpRet}$$

$$\frac{v_i : T_1 \to T_2 \qquad (D,\vec{\alpha}) \in \mathsf{ulpatt}(T_1) \qquad \vec{\alpha}\ \text{fresh}}{\langle A\,;M\,;\cdot\,;\cdot\,;\cdot\,;\vec{v}\rangle \xrightarrow{\underline{\mathsf{call}}(i,D[\vec{\alpha}])} \langle A \uplus \vec{\alpha}^{0,\cdot}\,;M\,;\cdot\,;\cdot\,;v_i\,D[\vec{\alpha}^0]\,;\cdot\rangle}\ \textsc{OpCallBarb}$$

$$\frac{\mathsf{next}_M(t) \subseteq \{\underline{\mathsf{call}}(i,D[\vec{\alpha}])\} \qquad K \neq \cdot \qquad v_i : T_1 \to T_2 \qquad (D,\vec{\alpha}) \in \mathsf{ulpatt}(T_1) \qquad t' = t + \underline{\mathsf{call}}(i,D[\vec{\alpha}])}{\langle A\,;M\,;K\,;t\,;V\,;\vec{v}\rangle \xrightarrow{\tau} \langle A \uplus \vec{\alpha}^{j,\vec{v}}\,;M[t']\,;K\,;t'\,;(v_i\,D[\vec{\alpha}^j])\,;\vec{v},V\rangle}\ \textsc{OpCall}$$

Fig. 2. The Labelled Transition System. We denote by $\cdot$ the empty stack, and by ε the empty sequence.

observed by the bisimulation moves. However, in rule PropRet this observation is *postponed*: the $\mathsf{ret}(D)$ move is appended to the current trace, and this trace is being stored in the M memory in the configuration. This memory will then be used to make distinctions between configurations in a bisimulation definition, when top-level transitions are reached. This storing of inner-level moves makes unobservable the order and repetition of proponent calls to opponent functions in the LTS, allowing to prove equivalences such as the one in Example 1.

Proponent Call:

Rule PropCall produces a transition when a call to an opponent abstract function $\alpha^j_{T_1\to T_2}$ is at reduction position in a proponent expression. Function $\mathsf{ulpatt}(v)$ is again used to decompose the call argument to context D and functions $\vec{v}$, whereas α^j is looked up in A to identify the knowledge $\vec{u}$ attached to this use of the α name at the time α^j was created. Then $\vec{v}$ and $\vec{u}$ are combined to create opponent's knowledge in the resulting configuration. The trace t accumulated in the (proponent) source configuration of the transition is being pushed onto the stack component K together with the continuation of the expression being reduced. This is because a proponent call transition triggers the creation of a new opponent-visible trace t', starting with the call move. This new trace is stored in the memory M and used in the resulting (opponent) configuration.

Segmentation of traces into opponent-visible trace fragments, as performed by this rule, is important for full abstraction of the NF bisimulation defined below. When configurations are compared by the bisimulation, the exact interleaving of these trace segments is not observable as the language is pure and opponent-generated functions have only a local view of the overall computation. Moreover, opponent-visible traces relate to O-views in game semantics but contain only a single (initiating) proponent call move.

Opponent Return:

An opponent configuration with a non-empty stack component K may return a value with rule OpRet. In order to obtain this value we extend ulpatt to the return type T through the use of symbolic function names: $\mathsf{ulpatt}(T)$ is the set of all pairs $(D, \vec{\alpha}_{\vec{T}})$ such that $\vdash D[\vec{\alpha}] : T$, where D is a value context that does not contain functions, and the types of $\vec{\alpha}$ and the corresponding holes match. Note that in this definition we leave the j annotation of α's blank as it is filled-in by the rule. In the resulting configuration $\alpha^{j,\vec{v}}$ is added in A, extending its domain by (α, j).

This transition can be performed in two cases; when:

- $\mathsf{next}_M(t) = \emptyset$. In this case the current opponent-visible trace t is not a strict prefix of a previously performed trace stored in M, and the configuration can non-deterministically perform this return transition. If it does, the resulting configuration stores in M the extended trace $t'' = t + \underline{\mathsf{ret}}(D[\vec{\alpha}])$. Note that j is not stored in moves and thus neither in M. Moreover in this case the $\vec{\alpha}$ used are chosen fresh, this is guaranteed by the implicit condition that M is legal and thus α cannot appear twice in M.
- $\mathsf{next}_M(t) = \{\underline{\mathsf{ret}}(D[\vec{\alpha}])\}$. In this case the current configuration is along an opponent-visible trace that has

occurred previously and performed a return as a next move. Thus because the opponent must have purely functional behaviour, the configuration can perform no other but this return transition.

If $\mathsf{next}_M(t)$ does not fall into one of the above cases the transition does not apply.

To encode functional behaviour, the current opponent knowledge $\vec{v}$ can only be stored in the abstract functions $\vec{\alpha}$ generated at this transition and stored in A. It *cannot* be carried forward otherwise in the resulting proponent function. Hence, if T' is a base type, this knowledge is lost after the transition.

Opponent Call:

The proponent function being called in these transitions defined by OPCALLBARB and OPCALL is one of those in the current opponent knowledge $\vec{v}$. We use the relative index i in $\vec{v}$ to refer to the function being called. The argument supplied to this function is obtained again by the function ulpatt applied to the argument type T_1.

Opponent call transitions are differentiated based on whether they are top- or inner-level. Top-level opponent calls (OPCALLBARB) are immediately observable and thus transitions are annotated with the move. Moreover, the opponent knowledge is dropped at the transition and not accumulated in the knowledge stack or created abstract function names. This is in line with applicative bisimulation where related top-level functions are called only at the point they become available in the bisimulation, and are provided with identical arguments, thus not not containing any related functions from the observer knowledge.

However inner-level opponent calls are not immediately observable and thus the corresponding move is stored in traces in M. As for inner opponent return transitions, $\mathsf{next}_M(t)$ may require that the transition must or cannot be applied.

Big-Step bisimulation:

Definition 10 (Trace transitions)**.** We use $\twoheadrightarrow$ for the reflexive and transitive closure of the $\xrightarrow{\tau}$ transition. We write $C \overset{\eta}{\twoheadrightarrow} C'$ to mean $C \twoheadrightarrow\xrightarrow{\eta}\twoheadrightarrow C'$, and $C \overset{t}{\twoheadrightarrow} C'$ to mean $C \overset{\eta}{\twoheadrightarrow}\overset{t'}{\twoheadrightarrow} C'$ when $t = \eta t'$. □

Note that, by definition, trace transitions derived by our LTS only contain $\underline{\mathsf{call}}$ and ret moves.

Definition 11. Given a closed expression $\vdash e : T$, the initial configuration associated to e is:

$$C_e = \langle \cdot\,; \cdot\,; \cdot\,; \cdot\,; e\,; \cdot \rangle$$

Accordingly, we can give the semantics of e as:

$$[\![e]\!] = \{(t, M) \mid \exists A, t', V, \vec{v}.\ C_e \overset{t}{\twoheadrightarrow} \langle A\,; M\,; \cdot\,; t'\,; V\,; \vec{v} \rangle\}.$$

□

A closed term e will be first evaluated by the LTS using the operational semantics rules (and PROPTAU). Once a value is reached, this will be communicated to the context by means of a proponent return (rule PROPRETBARB), after it has been appropriately decomposed. For there on, the game continues with opponent interrogating functions produced by proponent (using rule OPAPPBARB). Proponent can interrogate functions provided by opponent (PROPAPP), leading to further interaction all of which remains hidden (see τ-transitions), until proponent provides a return to opponent's top-level application (PROPRETBARB).

Example 12. We now revisit the terms in Example 1 to show how our LTS works. We start with term M_1 from Example 1 placed in an initial configuration $C_1 = \langle \cdot\,; \cdot\,; \cdot\,; \cdot\,; M_1\,; \cdot \rangle$. The first is a proponent return transition which moves the function into the opponent's knowledge.

$$C_1 \xrightarrow{\mathsf{ret}([])} \langle \cdot\,; \cdot\,; \cdot\,; \cdot\,; \cdot\,; M_1 \rangle = C_{12}$$

Then opponent calls and proponent immediately returns the second function (`fun g -> ...`), which we call M_{11}, and opponent calls M_{11}; all are top-level interactions.

$$\begin{aligned} C_{11} &\xrightarrow{\underline{\mathsf{call}}(1,\alpha_f)} \langle \alpha_f^0\,; \cdot\,; \cdot\,; \cdot\,; M_{11}[\alpha_f^0/\mathsf{f}]\,; \cdot \rangle \\ &\xrightarrow{\mathsf{ret}([])} \langle \alpha_f^0\,; \cdot\,; \cdot\,; \cdot\,; \cdot\,; M_{11}[\alpha_f^0/\mathsf{f}] \rangle \\ &\xrightarrow{\underline{\mathsf{call}}(1,\alpha_g)} \langle \alpha_f^0, \alpha_g^0\,; \cdot\,; \cdot\,; \cdot\,; M_{12}\,; \cdot \rangle = C_{12} \end{aligned}$$

where

$$\begin{aligned} M_{12} = {} & \texttt{if } \alpha_f^0\ \texttt{() == } \alpha_g^0\ \texttt{() then} \\ & \quad \texttt{if } \alpha_f^0\ \texttt{() == } \alpha_g^0\ \texttt{() then 0 else 1} \\ & \texttt{else 2} \end{aligned}$$

The following transition is a proponent call of α_f^0, followed (necessarily, due to types) by an opponent return.

$$\begin{aligned} C_{12} &\xrightarrow{\tau} \langle \alpha_f^0, \alpha_g^0\,; \{t_1\}\,; (\cdot, E_1)\,; t_1\,; \cdot\,; \cdot \rangle \\ &\qquad\text{(inner-level move } \mathsf{call}(\alpha_f, ()), \text{ rule PROPCALL)} \\ &\xrightarrow{\tau} \langle \alpha_f^0, \alpha_g^0\,; \{t_2\}\,; \cdot\,; \cdot\,; E_1[k_1]\,; \cdot \rangle = C_{13} \\ &\qquad\text{(inner-level move } \underline{\mathsf{ret}}(k_1), \text{ rule OPRET)} \end{aligned}$$

where $t_1 = \mathsf{call}(\alpha_f, ())$ and $t_2 = t_1, \underline{\mathsf{ret}}(k_1)$ and k_1 is an integer constant and $E_1 = (\texttt{if } [\cdot]_{\mathsf{int}} \texttt{ == } \alpha_g^0\ \texttt{() then ...})$. The transitions continue with the call and return of α_g.

$$\begin{aligned} C_{13} &\xrightarrow{\tau}\xrightarrow{\tau} \langle \alpha_f^0, \alpha_g^0\,; M_1\,; \cdot\,; \cdot\,; E_2[k_2]\,; \cdot \rangle = C_{14} \\ &\qquad\text{(inner-level moves } \mathsf{call}(\alpha_g, ()) \text{ and then } \underline{\mathsf{ret}}(k_2)) \end{aligned}$$

where $M_1 = \{t_2, t_3\}$ and $t_3 = \mathsf{call}(\alpha_f, ()), \underline{\mathsf{ret}}(k_1)$ and k_2 is an integer constant and $E_2 = (\texttt{if } k_1 \texttt{ == } [\cdot]_{\mathsf{int}} \texttt{ then ...})$.

The behaviour of α_f and α_g are now determined at this point from the traces t_2 and t_3 in the memory component of the configuration. Thus the following transitions only depend on whether $k_1 = k_2$. If they are not equal, proponent returns with a single $\mathsf{ret}(2)$ transition.

$$\begin{aligned} C_{14} &\xrightarrow{\mathsf{ret}(2)} \langle \alpha_f^0, \alpha_g^0\,; M_1\,; \cdot\,; \cdot\,; \cdot\,; \cdot \rangle = C_{15}(k_1, k_2) \\ &\qquad\text{(with } k_1 \neq k_2 \text{ in } M_1) \end{aligned}$$

If they are equal, the remaining transitions will be the following ones, reaching final configuration C_{15}.

$$C_{14} \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} \quad \text{(inner-level moves } \mathsf{call}(\alpha_f,()), \underline{\mathsf{ret}}(k_1), \mathsf{call}(\alpha_g,()), \underline{\mathsf{ret}}(k_2))$$
$$\xrightarrow{\mathsf{ret}(0)} \langle \alpha_f^0, \alpha_g^0 ; M_1 ; \cdot ; \cdot ; \cdot ; \cdot \rangle = C_{15}(k_1,k_2) \quad \text{(with } k_1 = k_2 \text{ in } M_1)$$

The $\underline{\mathsf{ret}}(k_1)$ and $\underline{\mathsf{ret}}(k_2)$ moves are the only possible at those points of the trace due to the memory component M_1, encoding a purely functional behaviour of the opponent.

Therefore, C_1 has only the following trace transitions:

$$C_1 \xrightarrow{\mathsf{ret}([])}\!\!\!\!\rightarrow \xrightarrow{\underline{\mathsf{call}}(1,\alpha_f)}\!\!\!\!\rightarrow \xrightarrow{\mathsf{ret}([])}\!\!\!\!\rightarrow \xrightarrow{\underline{\mathsf{call}}(1,\alpha_g)}\!\!\!\!\rightarrow \xrightarrow{\mathsf{ret}(2)}\!\!\!\!\rightarrow C_{15}(k_1,k_2) \quad \text{(with } k_1 \neq k_2)$$
$$C_1 \xrightarrow{\mathsf{ret}([])}\!\!\!\!\rightarrow \xrightarrow{\underline{\mathsf{call}}(1,\alpha_f)}\!\!\!\!\rightarrow \xrightarrow{\mathsf{ret}([])}\!\!\!\!\rightarrow \xrightarrow{\underline{\mathsf{call}}(1,\alpha_g)}\!\!\!\!\rightarrow \xrightarrow{\mathsf{ret}(0)}\!\!\!\!\rightarrow C_{15}(k_1,k_2) \quad \text{(with } k_1 = k_2)$$

Configuration $C_1' = \langle \cdot ; \cdot ; \cdot ; \cdot ; N_1 ; \cdot \rangle$ has the same trace transitions, with fewer inner call and return moves, in different order, resulting in top-level configurations with the same memory as the corresponding ones above. One can thus see that the two original terms M_1 and N_1 are equivalent under this LTS as $[\![M_1]\!] = [\![N_1]\!]$. □

Example 13. Here we explore the inequivalent terms from Example 3 to show how they are differentiated by our LTS. We focus on the configuration $C_2 = \langle \cdot ; \cdot ; \cdot ; \cdot ; M_2 ; \cdot \rangle$ and the transitions that differentiate it from N_2, corresponding to the behaviour of the context shown in Example 3.

To simplify notation in this example, we identify memory components with the same prefix closure and use in configurations below an instructive representative of each memory component equivalent class.

Since M_2 is a curried two-argument function similar to M_1 in the preceding example, we will have the following initial transitions.

$$C_2 \xrightarrow{\mathsf{ret}([])} \xrightarrow{\underline{\mathsf{call}}(1,\alpha_f)} \xrightarrow{\mathsf{ret}([])}$$
$$\xrightarrow{\underline{\mathsf{call}}(1,\mathsf{tt})}\!\!\!\!\rightarrow \langle \alpha_f^0 ; \cdot ; \cdot ; \cdot ; M_{21} ; \cdot \rangle = C_{21}$$

where $M_{21} = \alpha_f^0$ `(X, fun _ -> tt)` and `X` is the recursive function `fix X(d) ->` α_f^0 `(X, fun _ -> d)`. The next transition is an inner proponent call.

$$C_{21} \xrightarrow{\tau} \langle \alpha_f^0 ; \{t_1\} ; K_1 ; t_1 ; \cdot ; v_1, v_2 \rangle = C_{22} \quad \text{(inner move } \mathsf{call}(\alpha_f, ([],[])))$$

where $v_1 = \mathtt{X}$ and $v_2 =$ `fun _ -> tt` and $t_1 = \mathsf{call}(\alpha_f, ([],[]))$ and $K_1 = (\cdot, [\cdot])$. At this point of the interaction, opponent can either return a value or call one of the v_i. Since we are focusing on the behaviour of the discriminating context we show the following call to v_2:

$$C_{22} \xrightarrow{\tau}\!\!\!\!\rightarrow \langle \alpha_f^0 ; \{t_2\} ; K_1 ; t_2 ; \mathsf{tt} ; V_1 \rangle = C_{23} \quad \text{(inner move } \underline{\mathsf{call}}(2,\mathsf{ff}))$$

where the one-frame stack V_1 is (v_1, v_2) and $t_2 = t_1, \underline{\mathsf{call}}(2,\mathsf{ff})$. This is followed by

$$C_{23} \xrightarrow{\tau} \langle \alpha_f^0 ; \{t_3\} ; K_1 ; t_3 ; \cdot ; v_1, v_2 \rangle \quad \text{(inner } \mathsf{ret}(\mathsf{tt}))$$
$$\xrightarrow{\tau}\!\!\!\!\rightarrow \langle \alpha_f^0 ; \{t_4\} ; K_1 ; t_4 ; M_{22} ; V_1 \rangle = C_{24} \quad \text{(inner } \underline{\mathsf{call}}(1,\mathsf{ff}))$$

where $t_3 = t_2, \mathsf{ret}(\mathsf{tt})$ and $t_4 = t_3, \underline{\mathsf{call}}(1,\mathsf{ff})$ and $M_{22} = \alpha_f^0$ `(X, fun _ -> ff)`. Then the recursive call results to the transition:

$$C_{24} \xrightarrow{\tau} \langle \alpha_f^0 ; M_1 ; K_2 ; t_1 ; V_1 ; v_1, v_2' \rangle = C_{25} \quad \text{(inner } \mathsf{call}(\alpha_f, ([],[])))$$

where $M_1 = \{t_4\}[t_1] = \{t_4\}$ and $K_2 = (t_4, ([],[])), K_1$ and $v_2' =$ `fun _ -> ff`. At this point opponent must necessarily call v_2' as the current trace t_1 in the configuration is a prefix of t_4 in the memory component M_1 and $\mathsf{next}_M(t_1) = \underline{\mathsf{call}}(2,\mathsf{ff})$:

$$C_{25} \xrightarrow{\tau}\!\!\!\!\rightarrow \langle \alpha_f^0 ; M_2 ; K_2 ; t_2 ; \mathsf{ff} ; V_2 \rangle \quad \text{(inner } \underline{\mathsf{call}}(2,\mathsf{ff}))$$
$$\xrightarrow{\tau} \langle \alpha_f^0 ; M_3 ; K_2 ; t_3' ; V_1 ; v_1, v_2' \rangle = C_{26} \quad \text{(inner } \mathsf{ret}(\mathsf{ff}))$$

where $t_3' = t_2, \mathsf{ret}(\mathsf{ff})$ and $M_2 = \{t_4\}[t_2] = \{t_4\}$ and $M_3 = \{t_4\}[t_3'] = \{t_4, t_3'\}$ and $V_2 = (v_1, v_2'), V_1$. Trace t_3' is not a prefix of t_4 and therefore opponent can perform any move, including returning a value.

$$C_{26} \xrightarrow{\tau} \langle \alpha_f^0 ; M_4 ; K_1 ; t_4 ; \mathsf{tt} ; V_1 \rangle \quad \text{(inner } \underline{\mathsf{ret}}(\mathsf{tt}))$$
$$\xrightarrow{\tau} \langle \alpha_f^0 ; M_5 ; K_1 ; t_5 ; \cdot ; v_1, v_2 \rangle \quad \text{(inner } \mathsf{ret}(\mathsf{tt}))$$
$$\xrightarrow{\tau} \langle \alpha_f^0 ; M_6 ; \cdot ; \cdot ; \mathsf{tt} ; \cdot \rangle \quad \text{(inner } \underline{\mathsf{ret}}(\mathsf{tt}))$$
$$\xrightarrow{\mathsf{ret}(\mathsf{tt})} \langle \alpha_f^0 ; M_6 ; \cdot ; \cdot ; \cdot ; \cdot \rangle$$

where $M_4 = M_3[t_4'] = \{t_4, t_4'\}$ and $t_4' = t_3', \underline{\mathsf{ret}}(\mathsf{tt})$ and $M_5 = \{t_5, t_4'\}$ and $t_5 = t_4, \mathsf{ret}(\mathsf{tt})$ and $M_6 = \{t_6, t_4'\}$ and $t_6 = t_5, \underline{\mathsf{ret}}(\mathsf{tt})$.

Term N_2 is not able to perform this transition trace as once the first function of the pair is called, it diverges. □

We note that the LTS is deterministic at proponent configurations, but not at opponent configurations as the latter can fire more than one τ-transitions. Nonetheless, as the behaviour of opponent is restricted by the memory M, we can show the following.

Lemma 14 (M-determinacy)**.** *Given final configurations C, C_1, C_2 such that $C \xrightarrow{\underline{\mathsf{call}}(i,D[\vec{\alpha}])\,\mathsf{ret}(D')}\!\!\!\!\rightarrow C_i$ (for $i = 1, 2$), if $M_{C_1} \cup M_{C_2}$ is legal then $C_1 = C_2$.*

Proof. Let us set $M = M_{C_1} \cup M_{C_2}$. We break down the transitions from C to C_i as follows:

$$C \xrightarrow{\underline{\mathsf{call}}(i,D[\vec{\alpha}])} C_{i0} \xrightarrow{\tau} C_{i1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} C_{in_i} \xrightarrow{\mathsf{ret}(D')}\!\!\!\!\rightarrow C_i$$

We show that, for each $j = 0, \ldots, n$, where $n = \min(n_1, n_2)$, $C_{1j} = C_{2j}$. We do induction on j; the base case is clear from the LTS rules. Now consider $C_{ij} \xrightarrow{\tau} C_{i(j+1)}$. By IH, $C_{1j} = C_{2j}$. If they are (both) P-configurations then, by determinacy of proponent, the two configurations are bound to make the same move. Hence, $C_{1(j+1)} = C_{2(j+1)}$. If C_{ij} are

O-configurations, let their current (common) trace component be t. As the memory maps M_1, M_2 of $C_{1(j+1)}, C_{2(j+1)}$ are included in M, we must have $\mathsf{next}_{M_1}(t) = \mathsf{next}_{M_2}(t)$ and, hence, C_{1j} and C_{2j} must have made the same move and therefore $C_{1(j+1)} = C_{2(j+1)}$. Now, since $C_{1n} = C_{2n}$ and one of them makes a P-return then, by determinacy of proponent, the other must make the same P-return. Hence, $C_1 = C_2$. □

Weak bisimulation is defined in the standard way, albeit using the big-step transition relation corresponding to initial and final configurations.

Definition 15 (Weak (Bi-)Simulation)**.** A binary relation $\mathcal{R}$ on initial and final configurations is a *weak simulation* when for all $C_1 \mathrel{\mathcal{R}} C_2$:

- *Initial configurations:* if $C_1 \xrightarrow{\mathsf{ret}(D')} C_1'$, there exists C_2' such that $C_2 \xrightarrow{\mathsf{ret}(D')} C_2'$ and $C_1' \mathrel{\mathcal{R}} C_2'$;
- *Final configurations:* if $C_1 \xrightarrow{\mathsf{call}(i,D[\vec{\alpha}])\ \mathsf{ret}(D')} C_1'$ with $\vec{\alpha}$ fresh for C_2, there exists C_2' such that $C_2 \xrightarrow{\mathsf{call}(i,D[\vec{\alpha}])\ \mathsf{ret}(D')} C_2'$ and $C_1' \mathrel{\mathcal{R}} C_2'$;
- $M_{C_1} \subseteq M_{C_2}$ (where M_{C_i} is the M-component of C_i).

If $\mathcal{R}, \mathcal{R}^{-1}$ are weak simulations then $\mathcal{R}$ is a *weak bisimulation*. Similarity ($\lesssim$) and bisimilarity ($\approx$) are the largest weak simulation and bisimulation, respectively. □

This definition resembles that of applicative bisimulation for $\mathsf{PCF_v}$, in that related top-level functions applied to identical arguments must co-terminate and return related results. However the most important difference here is that there is no quantification over all possible programs. The context D is a value without any functions in it (essentially containing constants and/or pairs) which is determined by the type of the i'th function. The fresh names $\vec{\alpha}$ correspond to opponent-generated functions but are first-order entities that are equivalent up to renaming. Thus this definition constitutes a big-step Normal Form bisimulation.

Note in the previous definition the side-condition stipulating that the fresh abstract names used in the proponent application must not match any of the abstract names of C_2. That, along with the condition $M_{C_1} \subseteq M_{C_2}$, allow us to establish the following. Recall we write $\mathsf{an}(\chi)$ for the abstract names in χ.

Lemma 16. *Given weak simulation $\mathcal{R}$ and $C_1 \mathrel{\mathcal{R}} C_2$ with $\mathsf{an}(C_1) \subseteq \mathsf{an}(C_2)$, there is a weak simulation $\mathcal{R}' \subseteq \mathcal{R}$ such that $C_1 \mathrel{\mathcal{R}'} C_2$ and, for all $C_1' \mathrel{\mathcal{R}'} C_2'$, $\mathsf{an}(C_1') \subseteq \mathsf{an}(C_2')$.*

Proof. It suffices to show $\mathcal{R}' = \{(C_1, C_2) \in \mathcal{R} \mid \mathsf{an}(C_1) \subseteq \mathsf{an}(C_2)\}$ a weak simulation. Pick some $C_1 \mathrel{\mathcal{R}'} C_2$ and suppose $C_1 \xrightarrow{\mathsf{call}(i,D[\vec{\alpha}])\ \mathsf{ret}(D')} C_1'$ with $\vec{\alpha}$ fresh for C_2. By hypothesis, there is $C_2 \xrightarrow{\mathsf{call}(i,D[\vec{\alpha}])\ \mathsf{ret}(D')} C_2'$ with $C_1' \mathrel{\mathcal{R}} C_2'$. Therefore, $M_{C_1'} \subseteq M_{C_2'}$. Since $\mathsf{an}(C_1') = \mathsf{an}(C_1) \cup \mathsf{an}(M_{C_1'}) \cup \{\vec{\alpha}\}$, we can deduce that $\mathsf{an}(C_1') \subseteq \mathsf{an}(C_2')$. □

Definition 17 (Bisimilar Expressions)**.** Expressions $\vdash e_1 : T$ and $\vdash e_2 : T$ are bisimilar, written $e_1 \approx e_2$, when $C_{e_1} \approx C_{e_2}$. □

Lemma 18. $e_1 \approx e_2$ *iff* $[\![e_1]\!] = [\![e_2]\!]$.

Proof. Note first that if $e_1 \approx e_2$ and $(t, M) \in [\![e_1]\!]$ then, starting from C_{e_2}, we can simulate the transitions producing t and arrive at the same M (using also Lemma 16). Conversely, suppose that $[\![e_1]\!] = [\![e_2]\!]$ and define:

$$\mathcal{R} = \{(C_1, C_2) \mid M_{C_1} = M_{C_2} \wedge \exists t.\, C_{e_i} \xrightarrow{t} C_i \wedge C_i \text{ final}\}.$$

We show that $\mathcal{R}$ is a weak bisimulation. Suppose $C_1 \mathrel{\mathcal{R}} C_2$ with trace $C_{e_i} \xrightarrow{t} C_i$, and let $C_1 \xrightarrow{\mathsf{call}(i,D[\vec{\alpha}])\ \mathsf{ret}(D')} C_1'$ with $\vec{\alpha}$ fresh for C_2. As $[\![e_1]\!] = [\![e_2]\!]$, there is a transition sequence:

$$C_{e_2} \xrightarrow{t} \hat{C}_2 \xrightarrow{\mathsf{call}(i,D[\vec{\alpha}])\ \mathsf{ret}(D')} C_2'$$

such that $M_{C_1'} = M_{C_2'}$. Since $M_{C_2} = M_{C_1} \subseteq M_{C_1'}$, we have $M_{C_2} \subseteq M_{C_2'}$. Hence, starting from C_{e_2} and repeatedly applying Lemma 14, we conclude that $C_2 = \hat{C}_2$, and thus C_2 can match the challenge of C_1. Hence, $\mathcal{R}$ is a weak simulation and, by symmetry, a weak bisimulation. □

The previous result can be used to show that bisimilarity is sound and complete with respect to contextual equivalence. The proof is discussed in the next section.

Theorem 19 (Full abstraction)**.** $e_1 \approx e_2$ *iff* $e_1 \equiv e_2$.

Furthermore, normal-form similarity is fully abstract with respect to contextual approximation.

V. Full Abstraction

To prove that the LTS is sound and complete we use an extended LTS based on operational game semantics [19]. The latter differs from our main LTS in that proponent and opponent can play the same kinds of moves, and in particular they can pass fresh function names to the other player, or apply functions of the other player by referring to their corresponding names. This duality in roles allows for the modelling of both expressions and contexts. Moreover, all moves are recorded in the trace, not just top-level ones, which in turn enables us to compose two LTS's corresponding respectively to an expression and its context.

We shall call this the *game-LTS*, whereas the main LTS shall simply be *the/our LTS*. We shall be re-using some of our main LTS terminology here, for example traces will again be sequences of moves, albeit of different kind of moves. This is done for notional economy and we hope it is not confusing.

A. The game-LTS

We start by introducing an enriched notion of trace. Traces shall now consist of *moves* of the form:

$$\begin{array}{rrcl} \text{Moves} & m & ::= & p \mid o \\ \text{Proponent moves} & p & ::= & \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}]) \mid \mathsf{ret}(D[\vec{\mathfrak{p}}]) \\ \text{Opponent moves} & o & ::= & \underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}]) \mid \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}]) \end{array}$$

where $\mathfrak{o}, \mathfrak{p}$ (and variants thereof) are sourced from disjoint sets $ONames$ and $PNames$ of *opponent* and *proponent names* respectively. Names represent abstract functions and are used to abstract away the functions that a context and an expression are producing in a computation. We shall often be abbreviating "proponent" and "opponent" to P and O respectively and write, for instance, "O-moves" or "P-names".

A ***complete trace*** is then given by the following grammar.

$$\begin{aligned}
CT &\to CT_P \mid CT_O \\
CT_P &\to \mathsf{ret}(D[\vec{\mathfrak{p}}])\ CT_{OP} \\
CT_O &\to \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])\ CT_{PO} \\
CT_{OP} &\to \cdot \mid \underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])\ CT_{PO}\ \mathsf{ret}(D[\vec{\mathfrak{p}}])\ CT_{OP} \\
CT_{PO} &\to \cdot \mid \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])\ CT_{OP}\ \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])\ CT_{PO}
\end{aligned}$$

A *trace* is a prefix of a complete trace. A trace t is called *legal* if it satisfies these conditions:

- for each $t'p \sqsubseteq t$ with $p = \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])$ or $p = \mathsf{ret}(D)\vec{\mathfrak{p}}$:
  - $\vec{\mathfrak{p}}$ do not appear in t' — we say that move p *introduces* each $\mathfrak{p} \in \vec{\mathfrak{p}}_i$ — and
  - there is some move o' in t' that introduces $\mathfrak{o}$;
- for each $t'o \sqsubseteq t$ with $o = \underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$ or $o = \underline{\mathsf{ret}}(D)\vec{\mathfrak{o}}$:
  - $\vec{\mathfrak{o}}$ do not appear in t' — we say that move o *introduces* each $\mathfrak{o}_i \in \vec{\mathfrak{o}}$ — and
  - there is some move p' in t' that introduces $\mathfrak{p}$.

Thus, in a legal trace all applications refer to names introduced earlier in the trace. Put otherwise, all function calls must be to functions that are available when said calls are made. We say that an application $\underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$ (or $\mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])$) is *justified* by the (unique) earlier move that introduced $\mathfrak{p}$ (resp. $\mathfrak{o}$). On the other hand, a return is justified by the call to which it returns. In a legal trace, all call moves are justified.

Due to the modelled language being functional, not all names are visible to the players (i.e. proponent and opponent) at all times. For example if opponent makes two calls to proponent function $\mathfrak{p}$, say first $\underline{\mathsf{call}}(\mathfrak{p}, D_1[\vec{\mathfrak{o}}_1])$ and later $\underline{\mathsf{call}}(\mathfrak{p}, D_2[\vec{\mathfrak{o}}_2])$, the second call will hide from proponent all the trace related to the first one. This limitation is captured by the notion of *view*. Given a legal trace t, we define its *P-view* $\ulcorner t \urcorner$ and *O-view* $\llcorner t \lrcorner$ respectively as follows:

$$\ulcorner t \urcorner = \begin{cases} t & \text{if } |t| \leq 1 \\ \ulcorner t' \urcorner p & \text{if } t = t'p \\ \ulcorner t' \urcorner po & \text{if } t = t'pt''o \text{ and } o \text{ is justified by } p \end{cases}$$

$$\llcorner t \lrcorner = \begin{cases} t & \text{if } |t| \leq 1 \\ \llcorner t' \lrcorner o & \text{if } t = t'o \\ \llcorner t' \lrcorner op & \text{if } t = t'ot''p \text{ and } p \text{ is justified by } o \end{cases}$$

We will focus on traces where each player's moves are uniquely determined by their current view. This corresponds to game-semantics *innocence* (cf. [12]).

In the following definitions we employ basic elements from nominal set theory [33] to formally account for names in our constructions. Let us write $\mathcal{N}$ for $ONames \uplus PNames$. Finite-support name permutations that respect O- and P-ownership of names are given by:

$$\begin{aligned} Perm = \{\pi : \mathcal{N} \xrightarrow{\cong} \mathcal{N} \mid \exists X \subseteq_{\text{finite}} \mathcal{N}.\, \forall y \in \mathcal{N} \setminus X.\, \pi(y) = y \\ \wedge \forall x \in X.\, x \in ONames \iff \pi(x) \in ONames\} \end{aligned}$$

Given a trace t and a permutation π, we write $\pi \cdot t$ for the trace we obtain by applying π to all names in t. We write $t \sim t'$ if there exists some π such that $t' = \pi \cdot t$. The latter defines an equivalence relation, the classes of which we denote by $[t]$:

$$[t] = \{\pi \cdot t \mid \pi \in Perm\}.$$

Moreover, we define the sets of O-views and P-views of t (under permutation) as:

$$\begin{aligned} PV(t) &= \{\pi \cdot \ulcorner t' \urcorner \mid t' \sqsubseteq t \wedge \pi \in Perm\} \\ OV(t) &= \{\pi \cdot \llcorner t' \lrcorner \mid t' \sqsubseteq t \wedge \pi \in Perm\} \end{aligned}$$

Definition 20. A legal trace t is called a ***play*** if:

- for each $t'p, t''o \sqsubseteq t$, the justifier of p (of o) is included in $\ulcorner t' \urcorner$ (resp. $\llcorner t'' \lrcorner$);
- for all $t_1p_1, t_2p_2, t'_1o_1, t'_2o_2 \sqsubseteq t$,
  - if $\ulcorner t_1 \urcorner \sim \ulcorner t_2 \urcorner$ then $\ulcorner t_1p_1 \urcorner \sim \ulcorner t_2p_2 \urcorner$,
  - if $\llcorner t'_1 \lrcorner \sim \llcorner t'_2 \lrcorner$ then $\llcorner t'_1o_1 \lrcorner \sim \llcorner t'_2o_2 \lrcorner$.

We refer to the conditions above as *visibility* and *innocence* respectively. □

Visibility and innocence are standard game conditions (cf. [12], [27]): the former corresponds to the fact that an expression (or context) can only call functions in its syntactic context; while the latter enforces purely functional behaviour.

We can now proceed to the definition of the game-LTS. Similarly to the previous section, we extend the language syntax of Fig. 1 by including O-names as values. We define proponent and opponent *game-configurations* respectively by:

$$\langle \mathcal{A}\,;\kappa\,;K\,;t\,;e\,;V\,;\vec{\mathfrak{o}}\rangle \quad \text{and} \quad \langle \mathcal{A}\,;\kappa\,;K\,;t\,;V\,;\vec{\mathfrak{p}}\rangle$$

and range over them by $\mathcal{C}$ and variants. Here:

- $\mathcal{A}$ is a map which assigns to each (introduced) opponent name a sequence of proponent names. We write $\mathfrak{o}^{\vec{\mathfrak{p}}} \in \mathcal{A}$ for $\mathcal{A}(\mathfrak{o}) = \vec{\mathfrak{p}}$. The sequence $\vec{\mathfrak{p}}$ are the proponent (function) names that were available to opponent when the name $\mathfrak{o}$ was introduced.
- Dually, κ is a *concretion map* which assigns to each (introduced) proponent name the function that it represents and the opponent names that are available to it.
- t is a play recording all the moves that have been played thus far. Given t, we define the partial function $\mathsf{next}_O(t)$, which we use to impose innocence on O-moves, by:

  $$\mathsf{next}_O(t) = \{\pi{\cdot}o \mid \exists t'o \sqsubseteq t.\, \llcorner t \lrcorner = \pi{\cdot}\llcorner t' \lrcorner \wedge t(\pi{\cdot}o) \text{ a play}\}$$

  When we write $\mathsf{next}_O(t) \subseteq_\star [o]$, for some o, t, we mean that either $o \in \mathsf{next}_O(t)$ or $\mathsf{next}_O(t) = \emptyset$.
- K is a stack of proponent continuations (pairs of evaluation contexts and opponent names $\vec{\mathfrak{o}}$), and e is the expression reduced in proponent configurations.
- $\vec{\mathfrak{o}}$ and $\vec{\mathfrak{p}}$ are sequences of other-player names that are available to proponent and opponent respectively at the given point in the interaction; V is a stack of $\vec{\mathfrak{p}}$'s.

Note that we store the full trace in configurations and we use names ($\mathfrak{p}$ and variants) to abstract proponent higher-order values. There is no need of an M-component as we can rely

$$\frac{(D,\vec{v}) \in \mathsf{ulpatt}(v) \qquad t' = t + \mathsf{ret}(D[\vec{\mathfrak{p}}]) \qquad \kappa' = \kappa \uplus [\vec{\mathfrak{p}} \mapsto \vec{v}^{\vec{\mathfrak{o}}}]}{\langle \mathcal{A} ; \kappa ; K ; t ; v ; \vec{\mathfrak{p}}', V ; \vec{\mathfrak{o}} \rangle \xrightarrow{\mathsf{ret}(D[\vec{\mathfrak{p}}])} \langle \mathcal{A} ; \kappa' ; K ; t' ; V ; \vec{\mathfrak{p}}', \vec{\mathfrak{p}} \rangle} \text{ PROPRET}$$

$$\frac{e \to e'}{\langle \mathcal{A} ; \kappa ; K ; t ; e ; V ; \vec{\mathfrak{o}} \rangle \xrightarrow{\tau} \langle \mathcal{A} ; \kappa ; K ; t ; e' ; V ; \vec{\mathfrak{o}} \rangle} \text{ PROPTAU}$$

$$\frac{(D,\vec{v}) \in \mathsf{ulpatt}(v) \qquad t' = t + \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}]) \qquad \mathcal{A}(\mathfrak{o}) = \vec{\mathfrak{p}}' \qquad \kappa' = \kappa \uplus [\vec{\mathfrak{p}} \mapsto \vec{v}^{\vec{\mathfrak{o}}}]}{\langle \mathcal{A} ; \kappa ; K ; t ; E[\mathfrak{o}_{T_1 \to T_2}\, v] ; V ; \vec{\mathfrak{o}} \rangle \xrightarrow{\mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])} \langle \mathcal{A} ; \kappa' ; (E[\cdot]_{T_2}, \vec{\mathfrak{o}}), K ; t' ; V ; \vec{\mathfrak{p}}', \vec{\mathfrak{p}} \rangle} \text{ PROPAPP}$$

$$\frac{\mathsf{next}_O(t) \subseteq_\star [\underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])] \qquad (D,\vec{\mathfrak{o}}) \in \mathsf{ulpatt}(T') \qquad t' = t + \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])}{\langle \mathcal{A} ; \kappa ; (E[\cdot]_{T'}, \vec{\mathfrak{o}}'), K ; t ; V ; \vec{\mathfrak{p}} \rangle \xrightarrow{\underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])} \langle \mathcal{A} \uplus \vec{\mathfrak{o}}^{\vec{\mathfrak{p}}} ; \kappa ; K ; t' ; E[D[\vec{\mathfrak{o}}]] ; V ; \vec{\mathfrak{o}}', \vec{\mathfrak{o}} \rangle} \text{ OPRET}$$

$$\frac{\mathsf{next}_O(t) \subseteq_\star [\underline{\mathsf{call}}(\mathfrak{p}_i, D[\vec{\mathfrak{o}}])] \qquad \mathfrak{p}_i : T_1 \to T_2 \qquad \kappa(\mathfrak{p}_i) = v^{\vec{\mathfrak{o}}'} \qquad (D,\vec{\mathfrak{o}}) \in \mathsf{ulpatt}(T_1) \qquad t' = t + \underline{\mathsf{call}}(\mathfrak{p}_i, D[\vec{\mathfrak{o}}])}{\langle \mathcal{A} ; \kappa ; K ; t ; V ; \vec{\mathfrak{p}} \rangle \xrightarrow{\underline{\mathsf{call}}(\mathfrak{p}_i, D[\vec{\mathfrak{o}}])} \langle \mathcal{A} \uplus \vec{\mathfrak{o}}^{\vec{\mathfrak{p}}} ; \kappa ; K ; t' ; e ; \vec{\mathfrak{p}}, V ; \vec{\mathfrak{o}}', \vec{\mathfrak{o}} \rangle} \text{ OPAPP}$$

Fig. 3. The Game Labelled Transition System (game-LTS).

on the full play. We call a configuration *initial* if it is in one of these forms (called respectively *P- and O-initial*):[1]

$$\mathcal{C}_e = \langle \cdot ; \cdot ; \cdot ; \cdot ; e ; \varepsilon ; \varepsilon \rangle \text{ or } \mathcal{C}_E = \langle \cdot ; \cdot ; (E[\cdot]_T{:}\mathsf{unit}, \varepsilon) ; \cdot ; \cdot ; \varepsilon \rangle$$

and *final* if it is in one of these forms (*O- and resp. P-final*):

$$\langle \_ ; \_ ; \cdot ; \_ ; \cdot ; \_ \rangle \quad \text{or} \quad \langle \_ ; \_ ; \cdot ; \_ ; () ; \cdot ; \_ \rangle.$$

Note that, by definition of the LTS, a P-initial configuration can only lead to O-final configurations, whereas O-initial configurations lead to P-final configurations.

Definition 21. The game-LTS is defined by the rules in Fig. 3. Given initial configuration $\mathcal{C}$, we set:

$$CP(\mathcal{C}) = \{ t \in \mathsf{Pls}(\mathcal{C}) \mid t \text{ complete} \}$$

where we let $\mathsf{Pls}(\mathcal{C})$ be the set of plays produced by the LTS starting from $\mathcal{C}$. □

We can show that the traces produced by the game-LTS are plays and define a model for $\mathsf{PCF_v}$ based on sets of complete plays, but that would not be fully abstract. Though presented in operational form, our game-LTS is equivalent to the (base) game-model of $\mathsf{PCF_v}$ [10]. Consequently, if we model expressions by the sets of complete plays they produce, we miss even simple equivalences like $\lambda f.\, f() \equiv \lambda f.\, f(f())$ — plays are too intentional and do not take into account the limitations of functional contexts. To address this, one can use a semantic quotient (cf. [10]) or, alternatively, group the plays of an expression into sets of plays so as to profile functional contexts the expression may interact with (cf. [7]). Thus, an expression is modelled by a *set of sets of plays*, one for each possible context. We follow the latter approach, and also combine it with the fact that applicative tests suffice (cf. Proposition 7).

Definition 22. Given a P-starting play t, we call a move m of t ***top-level*** if:

- either m is the initial P-return of t,
- or m is an O-call justified by a top-level P-move,
- or m is a P-return to a top-level O-move.

We say that t is ***top-linear*** if each top-level O-move in t is justified by the P-move that precedes it. □

Hence, top-level moves are those that start from or go to a final configuration. If t is complete and top-linear then:

$$t = p_0 o_1 \cdots p_1 \cdots o_n \cdots p_n \quad \text{and} \quad \llcorner t \lrcorner = p_0 o_1 p_1 \cdots o_n p_n$$

where each o_{i+1} is justified by p_i, each p_i returns o_i ($i > 0$), and the o_i, p_i above are all the top-level moves in t. This means that, at the top level of a top-linear play, opponent may only choose one of the functions provided by proponent in their last move and examine it (i.e. call it), which precisely corresponds to what an applicative context would be able to do.

We can now present our main results for the game-LTS. Given initial P-configuration $\mathcal{C}$, we define:

$$OV(\mathcal{C}) = \{ OV(t) \mid t \in CP(\mathcal{C}) \}$$
$$OV_{tl}(\mathcal{C}) = \{ OV(t) \mid t \in CP(\mathcal{C}) \text{ and } t \text{ top-linear} \}$$

Proposition 23 (Correspondence)**.** *Given* $\vdash e_1, e_2 : T$, $OV_{tl}(\mathcal{C}_{e_1}) = OV_{tl}(\mathcal{C}_{e_2})$ *iff* $[\![e_1]\!] = [\![e_2]\!]$.

Proposition 24 (Game-LTS full abstraction)**.** *Given* $\vdash e_1, e_2 : T$, $e_1 \equiv e_2$ *iff* $OV_{tl}(\mathcal{C}_{e_1}) = OV_{tl}(\mathcal{C}_{e_2})$.

Theorem 19 follows from the two results above. For the first result we build a translation from the game-LTS to the (plain) LTS that forms a certain bisimulation between the two systems. To prove full abstraction of the game-LTS we use standard and operational game semantics techniques (cf. [12],

[1] we write $V = \cdot$ for an empty stack, and $V = \varepsilon$ for a singleton stack containing the empty sequence; moreover, here and elsewhere, we use underscore (_) to denote any component of the appropriate type.

[19], [8]) along with the characterisation of PCF equivalence by sets of O-views presented in [7].

VI. Prototype Implementation

We implemented the LTS with symbolic higher-order transitions in a prototype bisimulation checking tool for programs written in an ML-like syntax for $\mathrm{PCF_v}$. Our tool implements a Bounded Symbolic Execution–via calls to Z3–for a big-step bisimulation of the LTS; the tool was developed in OCaml[2].

The tool performs symbolic execution of base type values through an extension of the LTS to include a *symbolic environment* $\sigma : \mathsf{Val} \rightarrow \mathsf{Val}$ that accumulates constraints on *symbolic constants* $\varkappa \in \mathsf{Val}$ that extend the set of values. Symbolic constants are of base type and may only be introduced by opponent moves (arguments and return values) and by reducing expressions that involve symbolic constants; their semantics follows standard symbolic execution. The exploration is performed over *configuration pairs* $\langle C_1, C_2, M, \sigma, k\rangle$ of bisimilar term configurations C_1 and C_2, shared memory M and given bound k. This shared memory is the combination of memories in C_1 and C_2. When configurations C_1 and C_2 are final, equivalence requires $M_{C_1} = M_{C_2} = M$. Being a symbolic execution tool, our prototype implementation is *sound* (reports only true positives and true negatives) and *bounded-complete* since it exhaustively and precisely explores all possible paths up to the given bound, which defines the number of consecutive function calls allowed.

Because of the infinite nature of proving equivalence — and even of disproving equivalence — of pure higher-order programs, a bounded exploration often does not suffice for automatic verification. For this reason, we implement simple enhancements that attempt to prune the state-space and/or prove that cycles have been reached to finitise the exploration for several examples in our testsuite. We currently have not implemented more involved up-to bisimulation enhancements, perhaps guided by user annotations, which we leave for future work. In particular we make use of:

- *Memoisation*, which caches configuration pairs. When bounded exploration reaches a memoised configuration pair, the tool does not explore any further outgoing transitions from this pair; these were explored already when the pair was added to the memoisation set.
- *Identity*, which deems two configurations in a pair equivalent when they are syntactically identical; no further exploration is needed in this case.
- *Normalisation*, which renames bound variables and symbolic constants before comparing configuration pairs for membership in the memoisation set. This also normalises the symbolic environments σ in the configuration pairs.
- *Proponent call caching*, which caches proponent calls once the corresponding opponent return is reached. When the same call (same function applied to the same argument) is reached again on the same trace, it is immediately followed by the cached opponent return move. Performing this second call would not have materially changed the configuration, as the behaviour of the call is determined by the traces in the memory M of the configuration.
- *Opponent call skipping*, which caches opponent calls once the corresponding proponent return is reached. If the same call is possible from later configurations with the same opponent knowledge, the call is skipped as the opponent cannot increase its knowledge by repeating the same call.
- *Stack-based loop detection*, which searches the stack component K of a configuration for nested identical proponent calls. When this happens, it means that the configuration is on an infinite trace of interactions between opponent and proponent which will keep applying the same function indefinitely. We deem these configurations diverging.

[2]https://github.com/LaifsV1/pcfeq

Running our tool on the examples in this paper on an Intel Core i7 1.90GHz machine with 32GB RAM running OCaml 4.10.0 and Z3 4.8.10 on Ubuntu 20.04 we obtain the following three-trial average results: Example 1, deemed equivalent, 8ms; Example 2, inequivalent, 3ms; Example 3, inequivalent, 4ms; 4, inequivalent, 3ms. For the entire benchmark of thirty seven program pairs, we successfully verify six equivalences and nineteen inequivalences with twelve inconclusive results in 471ms total time. The complete set of examples is available in our online repository.

VII. Conclusion

We have proposed a technique which combines operational and denotational approaches in order to provide a (quotient-free) characterisation of contextual equivalence in call-by-value PCF. This technique provides the first fully abstract normal form bisimulation for this language. We have justified several of our choices in designing our LTS via examples, and we believe the LTS is succinct in not carrying more information than needed for completeness. Our technique gives rise to a sound and complete technique for checking of $\mathrm{PCF_v}$ program equivalence, which we implemented into a bounded bisimulation checking tool.

After testing our tool implementation, we have found it useful for deciding instances of the equivalence problem. This is particularly true for inequivalences: the tool was able to verify most of our examples, including some which were difficult to reason about even informally. Further testing and optimisation of the implementation are needed in order to assess its practical relevance, particularly on larger examples. Currently, the main limitation for the tool is the difficulty in establishing equivalences, as these typically entail infinite bisimulations and are hard to capture in a bounded manner. To address this, we aim to develop up-to techniques [34] and (possibly semi-automatic) abstraction methods in order to finitise the examined bisimulation space.

References

[1] Abramsky, S.: The Lazy Lambda Calculus, p. 65–116. Addison-Wesley Longman Publishing Co., Inc., USA (1990)

[2] Abramsky, S., Malacaria, P., Jagadeesan, R.: Full abstraction for PCF. In: Hagiya, M., Mitchell, J.C. (eds.) Theoretical Aspects of Computer

Software, International Conference TACS '94. LNCS, vol. 789, pp. 1–15. Springer (1994)

[3] Abramsky, S., McCusker, G.: Call-by-value games. In: Nielsen, M., Thomas, W. (eds.) Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference of the EACSL. LNCS, vol. 1414, pp. 1–17. Springer (1997)

[4] Appel, A.W., McAllester, D.: An indexed model of recursive types for foundational proof-carrying code. ACM Trans. Program. Lang. Syst. **23**(5), 657–683 (sep 2001)

[5] Biernacki, D., Lenglet, S.: Normal form bisimulations for delimited-control operators. In: Schrijvers, T., Thiemann, P. (eds.) Functional and Logic Programming. pp. 47–61. Springer, Berlin, Heidelberg (2012)

[6] Biernacki, D., Lenglet, S., Polesiuk, P.: A complete normal-form bisimilarity for state. In: FOSSACS 2019, Held as Part of ETAPS 2019. Springer (2019)

[7] Churchill, M., Laird, J., McCusker, G.: A concrete representation of observational equivalence for PCF (2010), http://arxiv.org/abs/1003.0107

[8] Ghica, D.R., Tzevelekos, N.: A system-level game semantics. In: Berger, U., Mislove, M.W. (eds.) Proceedings of the 28th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2012. Electronic Notes in Theoretical Computer Science, vol. 286, pp. 191–211. Elsevier (2012)

[9] Gordon, A.D.: Functional programming and input/output. Ph.D. thesis, University of Cambridge, UK (1992)

[10] Honda, K., Yoshida, N.: Game-theoretic analysis of call-by-value computation. Theor. Comput. Sci. **221**(1-2), 393–456 (1999)

[11] Howe, D.J.: Proving congruence of bisimulation in functional programming languages. Information and Computation **124**(2), 103–112 (1996)

[12] Hyland, J.M.E., Ong, C.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. **163**(2), 285–408 (2000)

[13] Jaber, G.: Operational nominal game semantics. In: Pitts, A.M. (ed.) Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of ETAPS 2015. LNCS, vol. 9034, pp. 264–278. Springer (2015)

[14] Jagadeesan, R., Pitcher, C., Riely, J.: Open bisimulation for aspects. In: Rashid, A., Ossher, H. (eds.) Transactions on Aspect-Oriented Software Development V. pp. 72–132. Springer, Berlin, Heidelberg (2009)

[15] Koutavas, V., Levy, P.B., Sumii, E.: From applicative to environmental bisimulation. In: Mislove, M.W., Ouaknine, J. (eds.) Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011. Electronic Notes in Theoretical Computer Science, vol. 276, pp. 215–235. Elsevier (2011)

[16] Koutavas, V., Lin, Y.Y., Tzevelekos, N.: From bounded checking to verification of equivalence via symbolic up-to techniques. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 178–195. Springer International Publishing, Cham (2022)

[17] Koutavas, V., Wand, M.: Bisimulations for untyped imperative objects. In: Sestoft, P. (ed.) ESOP 2006, Held as Part of ETAPS 2006. LNCS, vol. 3924, pp. 146–161. Springer (2006)

[18] Koutavas, V., Wand, M.: Small bisimulations for reasoning about higher-order imperative programs. In: Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006. ACM (2006)

[19] Laird, J.: A fully abstract trace semantics for general references. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) Automata, Languages and Programming. pp. 667–679. Springer, Berlin, Heidelberg (2007)

[20] Lassen, S.B., Levy, P.B.: Typed normal form bisimulation for parametric polymorphism. In: LICS. pp. 341–352. IEEE Computer Society (2008)

[21] Lassen, S.: Bisimulation in untyped lambda calculus:: Böhm trees and bisimulation up to context. In: MFPS XV, Mathematical Foundations of Progamming Semantics, Fifteenth Conference. vol. 20, pp. 346–374 (1999)

[22] Lassen, S.: Eager normal form bisimulation. In: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science. p. 345–354. LICS '05, IEEE Computer Society, USA (2005)

[23] Lassen, S.B.: Normal form simulation for McCarthy's Amb. In: Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI). vol. 155, pp. 445–465 (2006)

[24] Lassen, S.B., Levy, P.B.: Typed normal form bisimulation. In: Computer Science Logic. Springer, Berlin, Heidelberg (2007)

[25] Lassen, S.B., Levy, P.B.: Typed normal form bisimulation for parametric polymorphism. In: 2008 23rd Annual IEEE Symposium on Logic in Computer Science. pp. 341–352 (2008)

[26] Loader, R.: Finitary PCF is not decidable. Theor. Comput. Sci. **266**(1-2), 341–364 (2001)

[27] McCusker, G.: Games and full abstraction for a functional metalanguage with recursive types. CPHC/BCS distinguished dissertations, Springer (1998)

[28] Milner, R.: Fully abstract models of typed *lambda*-calculi. Theor. Comput. Sci. **4**(1), 1–22 (1977)

[29] Morris, Jr., J.H.: Lambda Calculus Models of Programming Languages. Ph.D. thesis, MIT, Cambridge, MA (1968)

[30] Nickau, H.: Hereditarily sequential functionals. In: Nerode, A., Matiyasevich, Y.V. (eds.) Logical Foundations of Computer Science, Third International Symposium, LFCS'94, St. Petersburg, Russia, July 11-14, 1994, Proceedings. LNCS, vol. 813, pp. 253–264. Springer (1994)

[31] O'Hearn, P., Riecke, J.: Kripke logical relations and PCF. Information and Computation **120**(1), 107–116 (1995)

[32] Pitts, A.M.: Existential types: Logical relations and operational equivalence. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) Automata, Languages and Programming. pp. 309–326. Springer (1998)

[33] Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press, New York, NY, USA (2013)

[34] Pous, D., Sangiorgi, D.: Enhancements of the bisimulation proof method. In: Advanced Topics in Bisimulation and Coinduction. CUP (2012)

[35] Sangiorgi, D.: The lazy lambda calculus in a concurrency scenario. Information and Computation **111**(1), 120–153 (1994)

[36] Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. ACM Trans. Program. Lang. Syst. **33**(1), 5:1–5:69 (2011)

[37] Støvring, K., Lassen, S.B.: A complete, co-inductive syntactic theory of sequential control and state. In: Hofmann, M., Felleisen, M. (eds.) Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007. pp. 161–172. ACM (2007)

[38] Sumii, E., Pierce, B.C.: A bisimulation for dynamic sealing. Theor. Comput. Sci. **375**(1-3), 169–192 (2007)

[39] Sumii, E., Pierce, B.C.: A bisimulation for type abstraction and recursion. J. ACM **54**(5), 26 (2007)

[40] Tzevelekos, N.: Full abstraction for nominal general references. Log. Methods Comput. Sci. **5**(3) (2009)

Appendix
Soundness and completeness

A. Soundness of game-LTS

In this section we show that the game-LTS is sound for contextual equivalence. Our argument proceeds by first introducing two notions of composition on the LTS (so called *internal* and *external*) and then by using compositionality to relate the semantics of an expression-in-context to traces of the expression and the context respectively (cf. [19]).

We start off with a definition and a couple of technical lemma on the game-LTS. If $\mathcal{C}$ is an O-configuration, we also define:

$$CP^{+}(\mathcal{C}) = \{t\,\mathsf{ret}(()) \mid \mathcal{C} \xrightarrow{t}\!\!\!\!\rightarrow \langle \_\,;\_\,;\cdot\,;\_\,;()\,;\cdot\,;\_\rangle\}.$$

where note above that t must be complete.

Lemma 25. *Let t be a play. Then both $\ulcorner t \urcorner$ and $\llcorner t \lrcorner$ are legal traces satisfying visibility.*

Proof. Proved using standard game semantics techniques (e.g. [27]). □

Lemma 26. *Let $\mathcal{C}$ be an initial configuration and suppose that $\mathcal{C} \xrightarrow{t}\!\!\!\!\rightarrow \mathcal{C}'$. Then:*

1) *t is a legal trace and if $\mathcal{C}'$ has components $\mathcal{A}, \kappa$ then the names in t are precisely $\mathcal{A} \cup \mathsf{dom}(\kappa)$;*
2) *for any name permutation π, $\mathcal{C} \xrightarrow{\pi\cdot t}\!\!\!\!\rightarrow \pi \cdot \mathcal{C}'$;*
3) *if $\mathcal{C}' = \langle \_\,;\_\,;\_\,;t\,;\_\,;\vec{\mathfrak{p}}\rangle$ then the P-names in $\llcorner t \lrcorner$ are $\vec{\mathfrak{p}}$ (in the same order);*
4) *if $\mathcal{C}' = \langle \_\,;\_\,;\_\,;t\,;\_\,;\_\,;\vec{\mathfrak{o}}\rangle$ then the O-names in $\ulcorner t \urcorner$ are $\vec{\mathfrak{o}}$ (in the same order); moreover, for each play $t'o$ we have $\mathcal{C}' \xrightarrow{o} \mathcal{C}''$ for some $\mathcal{C}''$;*
5) *if $\mathcal{C} \xrightarrow{\mathcal{C}}\!\!\!\!\rightarrow''$ with $\ulcorner t \urcorner = \ulcorner t' \urcorner$ and both $\mathcal{C}', \mathcal{C}''$ are P-configurations not preceded by P-configurations then the expressions of $\mathcal{C}', \mathcal{C}''$ are the same;*
6) *t is a play;*
7) *if, for some play $\hat{t}$, $PV(t) = PV(\hat{t})$, then $\mathcal{C} \xrightarrow{\hat{t}}\!\!\!\!\rightarrow \mathcal{C}''$ for some $\mathcal{C}''$.*

Proof. For 1, the fact that t is a legal trace follows by construction of the game-LTS, i.e. by the stack discipline and the fact that all names played as arguments are fresh. Moreover, $\mathcal{A}$ and κ are populated precisely by names introduced in the trace t.

Claim 2 follows by nominal sets reasoning.

For 3, we proceed by induction on $|t|$. The case for $t \leq 1$ is straightforward. Otherwise, suppose that the last move in t is some $p = \mathsf{ret}(\_[\_])$ and $|t| > 1$. By the stack discipline (for the V component), we have that the current V component is some $\vec{\mathfrak{p}}, V'$, and that p is returning the O-application that pushed $\vec{\mathfrak{p}}$. By IH, the names in the O-view at that O-application were $\vec{\mathfrak{p}}$, which suffices for our claim. Finally, suppose that the last move in t is some $p = \mathsf{call}(\mathfrak{o}, \_[\_])$ and suppose that $\mathfrak{o}$ in this application is decorated as $\mathfrak{o}^{\vec{\mathfrak{p}}'}$. By IH, at the point of introduction of $\mathfrak{o}$ in t, the O-view contained precisely $\vec{\mathfrak{p}}'$, which concludes the claim.

For 4, we proceed by induction on $|t|$. The case for $t \leq 1$ is straightforward. Otherwise, suppose that the last move in t is some $o = \underline{\mathsf{ret}}(\_[\_])$ and $|t| > 1$. By the stack discipline (for the K component), we have that the current K component is some $(E, \vec{\mathfrak{o}}), K'$, and that o is returning the P-application that pushed $(E, \vec{\mathfrak{o}})$. By IH, the names in the P-view at that P-application were $\vec{\mathfrak{o}}$, which suffices for our claim. Now suppose that the last move in t is some $o = \underline{\mathsf{call}}(\mathfrak{p}, \_[\_])$ and suppose that $\mathfrak{p}$ is mapped to some value decorated as $v^{\vec{\mathfrak{o}}'}$. By IH, at the point of introduction of $\mathfrak{p}$ in t, the P-view contained precisely $\vec{\mathfrak{o}}'$, which concludes the claim. Finally, if $t'o$ is a play then o satisfies the next_O requirements, and if o is an application of some P-name $\mathfrak{p}$ then the latter is in the last component of $\mathcal{C}'$. Hence $\mathcal{C}'$ can play o.

For 5, we do induction on $n = |\ulcorner t \urcorner|$. The cases for $n \leq 1$ are straightforward. Suppose that the last move in t, t' is some $o = \underline{\mathsf{ret}}(\_[\_])$ and $|\ulcorner t' \urcorner| > 1$. By the stack discipline (for the K component), we have that the current K component is some $(E, \vec{\mathfrak{o}}), K'$, and that o is returning the P-application that pushed $(E, \vec{\mathfrak{o}})$. Using the IH, and applying Tau rules on the same expression on both sides, we have that the expression $E[\mathfrak{o}v]$ triggering that P-application is common in the two cases. Hence, the expressions in $\mathcal{C}', \mathcal{C}''$ are the same. Finally, suppose that the last move in t, t' is some $o = \underline{\mathsf{call}}(\mathfrak{p}, \_[\_])$ and suppose that $\mathfrak{p}$ is mapped to values v, v' in $\mathcal{C}', \mathcal{C}''$ respectively. By IH (and also applying Tau rules), at the point of introduction of $\mathfrak{p}$ there was a common expression in both traces and, therefore, $v = v'$. This concludes the claim.

For 6, by claims 3 and 4 we have that applications refer to names in the corresponding views. Moreover, O-innocence follows by definition of the $\mathsf{next}_O(\cdot)$ function. We need to show P-innocence. Suppose that $t_1p_1, t_2p_2 \sqsubseteq t$ with $\ulcorner t_1 \urcorner = \pi \cdot \ulcorner t_2 \urcorner$ for some π. Let $\mathcal{C} \xrightarrow{t_1}\!\!\!\!\rightarrow \mathcal{C}_1$ and $\mathcal{C} \xrightarrow{t_2}\!\!\!\!\rightarrow \mathcal{C}_2$, with $\mathcal{C}_1, \mathcal{C}_2$ not preceded by P-configurations, be the corresponding LTS reductions. By 2, $\mathcal{C} \xrightarrow{\pi\cdot t_2}\!\!\!\!\rightarrow \pi \cdot \mathcal{C}_2$ and hence, by 5, $\mathcal{C}_1$ and $\pi \cdot \mathcal{C}_2$ contain the same expressions. Thus, the Tau rules that can be applied on $\mathcal{C}_1$ and $\mathcal{C}_2$ are uniquely defined, and reach to configurations $\mathcal{C}_1', \mathcal{C}_2'$ such that $\mathcal{C}_1'$ and $\pi \cdot \mathcal{C}_2'$ contain the same expressions and are ready to trigger the same P-move, apart possibly from the selection of fresh P-names. By strong support lemma, we obtain that $\ulcorner t_1p_1 \urcorner \sim \ulcorner t_2p_2 \urcorner$.

For 7, we show that $\mathcal{C}$ produces every prefix $t' \sqsubseteq \hat{t}$, by induction on $|t'|$. If $|t'| \leq 1$ then by hypothesis we have that $\mathcal{C}$ produces t'. Suppose now $t' = t''p$ with t'' non-empty. By IH, $\mathcal{C} \xrightarrow{t''}\!\!\!\!\rightarrow \mathcal{C}''$, and suppose that $\mathcal{C}''$ is not preceded by a P-configuration. By hypothesis, there are a π and a prefix $\tilde{t}$ of $\pi \cdot t$ such that $\ulcorner\tilde{t}\urcorner = \ulcorner t''\urcorner$, and say $\mathcal{C} \xrightarrow{\tilde{t}}\!\!\!\!\rightarrow \tilde{\mathcal{C}}$, with the latter not preceded by a P-configuration. Then, by 5, $\mathcal{C}''$ and $\tilde{\mathcal{C}}$ contain the same expression and, hence, will play the same move p, assuming an appropriate choice of π, as required. Finally, suppose $t' = t''o$ with t'' non-empty. By IH, $\mathcal{C} \xrightarrow{t''}\!\!\!\!\rightarrow \mathcal{C}''$ and $\mathcal{C}''$ is an O-configuration. By 4, we have that $\mathcal{C}'' \xrightarrow{o} \mathcal{C}'''$, as required. □

1) Semantic Composition

We start by defining a notion of composition that combines the traces produced by two configurations. These are supposed to correspond to an expression and its context, but for now we will only require that the configurations satisfy a set of compatibility conditions.

Let ϕ be a partial bijection from *ONames* to *PNames*. Given a trace t with its names included in $\mathsf{dom}(\phi) \cup \mathsf{rng}(\phi)$, we define its *dual* with respect to ϕ, written $\bar{t}^\phi$, inductively by $\bar{\varepsilon}^\phi = \varepsilon$ and:

$$\overline{\mathsf{ret}(D[\vec{\mathfrak{p}}])}^\phi = \underline{\mathsf{ret}}(D[\phi^{-1}(\vec{\mathfrak{p}})]), \qquad \overline{\underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])}^\phi = \mathsf{ret}(D[\phi(\vec{\mathfrak{o}})])$$

$$\overline{\mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])}^\phi = \underline{\mathsf{call}}(\phi(\mathfrak{o}), D[\phi^{-1}(\vec{\mathfrak{p}})]) \qquad \overline{\underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])}^\phi = \mathsf{call}(\phi^{-1}(\mathfrak{p}), D[\phi(\vec{\mathfrak{o}})]).$$

We say configurations $\mathcal{C}$ and $\mathcal{C}'$ of opposite polarity are *ϕ-compatible* ($\mathcal{C} \asymp_\phi \mathcal{C}'$) if:

- their names are disjoint: $\mathcal{A} \cap \mathcal{A}' = \mathsf{dom}(\kappa) \cap \mathsf{dom}(\kappa') = \emptyset$;
- their traces are complementary up to ϕ, i.e. $t = \bar{t'}^\phi$;
- their stacks are compatible, written $(K, V) \asymp_\phi (K', V')$, which means:
  - $K = K' = V = V' = \varepsilon$; or
  - $K = (E, \vec{\mathfrak{o}}), K_1$ and $V' = \vec{\mathfrak{p}}, V_1'$, with $\phi(\vec{\mathfrak{o}}) = \vec{\mathfrak{p}}$, and $(K_1, V) \asymp_{\phi'} (K', V_1')$ with $\phi' \subseteq \phi$; or
  - $V = \vec{\mathfrak{p}}, V_1$ and $K' = (E, \vec{\mathfrak{o}}), K_1'$, with $\phi(\vec{\mathfrak{o}}) = \vec{\mathfrak{p}}$, and $(K, V_1) \asymp_{\phi'} (K_1', V')$ with $\phi' \subseteq \phi$;
- $\phi : \mathcal{A} \cup \mathcal{A}' \to \mathsf{dom}(\kappa) \cup \mathsf{dom}(\kappa')$ and:
  - for each $\mathfrak{o}^{\vec{\mathfrak{p}}} \in \mathcal{A}_1$, if $\kappa_2(\phi(\mathfrak{o})) = v^{\vec{\mathfrak{o}}}$ then $\phi(\vec{\mathfrak{o}}) = \vec{\mathfrak{p}}$;
  - for each $\mathfrak{o}^{\vec{\mathfrak{p}}} \in \mathcal{A}_2$, if $\kappa_1(\phi(\mathfrak{o})) = v^{\vec{\mathfrak{o}}}$ then $\phi(\vec{\mathfrak{o}}) = \vec{\mathfrak{p}}$;
  - if $\mathcal{C}_1, \mathcal{C}_2$ have last components $\vec{\mathfrak{o}}, \vec{\mathfrak{p}}$ (or $\vec{\mathfrak{p}}, \vec{\mathfrak{o}}$) then $\phi(\vec{\mathfrak{o}}) = \vec{\mathfrak{p}}$.

With these definitions, we proceed to defining different notions of composition.

Suppose $\mathcal{C} \asymp_\phi \mathcal{C}'$. The following rules define the semantic composition of two configurations (symmetric rules $\textsc{App}_R, \textsc{Ret}_R$ are omitted).

$$\frac{\mathcal{C}_1 \xrightarrow{\tau} \mathcal{C}_1' \qquad \mathcal{C}_2' = \mathcal{C}_2}{\mathcal{C}_1 \oslash_\phi \mathcal{C}_2 \to' \mathcal{C}_1' \oslash_\phi \mathcal{C}_2'}\ \textsc{Int}_L \qquad \frac{\mathcal{C}_2 \xrightarrow{\tau} \mathcal{C}_2' \qquad \mathcal{C}_1' = \mathcal{C}_1}{\mathcal{C}_1 \oslash_\phi \mathcal{C}_2 \to' \mathcal{C}_1' \oslash_\phi \mathcal{C}_2'}\ \textsc{Int}_R$$

$$\frac{\mathcal{C}_1 \xrightarrow{\mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])} \mathcal{C}_1' \qquad \mathcal{C}_2 \xrightarrow{\underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])} \mathcal{C}_2' \qquad \phi(\mathfrak{o}) = \mathfrak{p} \qquad \vec{\mathfrak{o}} \notin \mathcal{C}_1 \wedge \vec{\mathfrak{p}} \notin \mathcal{C}_2}{\mathcal{C}_1 \oslash_\phi \mathcal{C}_2 \to' \mathcal{C}_1' \oslash_{\phi \uplus [\vec{\mathfrak{o}} \mapsto \vec{\mathfrak{p}}]} \mathcal{C}_2'}\ \textsc{App}_L$$

$$\frac{\mathcal{C}_1 \xrightarrow{\mathsf{ret}(D)\vec{\mathfrak{p}}} \mathcal{C}_1' \qquad \mathcal{C}_2 \xrightarrow{\underline{\mathsf{ret}}(D)\vec{\mathfrak{o}}} \mathcal{C}_2' \qquad \vec{\mathfrak{o}} \notin \mathcal{C}_1 \wedge \vec{\mathfrak{p}} \notin \mathcal{C}_2}{\mathcal{C}_1 \oslash_\phi \mathcal{C}_2 \to' \mathcal{C}_1' \oslash_{\phi \uplus [\vec{\mathfrak{o}} \mapsto \vec{\mathfrak{p}}]} \mathcal{C}_2'}\ \textsc{Ret}_L$$

We can show the following.

Lemma 27. *If $\mathcal{C}_1 \asymp_\phi \mathcal{C}_2$ and $\mathcal{C}_1 \oslash_\phi \mathcal{C}_2 \to' \mathcal{C}_1' \oslash_{\phi'} \mathcal{C}_2'$ then $\mathcal{C}_1' \asymp_{\phi'} \mathcal{C}_2'$.*

2) Composite Semantics and Internal Composition

We now introduce the notion of composing LTS configurations *internally*, which occurs when merging two compatible configurations into a single expression. The composition will ensure that the external functions (i.e. O-names) of each configuration are closed by the other one. Hence, we will use extended configurations of the form:

$$(\mathcal{A}_L, \mathcal{A}_R, \kappa_L, \kappa_R, \phi, e) \text{ or, for brevity, } (\vec{\mathcal{A}}, \vec{\kappa}, \phi, e)$$

such that $\mathcal{A}_L \cap \mathcal{A}_R = \mathsf{dom}(\kappa_L) \cap \mathsf{dom}(\kappa_R) = \emptyset$ and $\phi : \mathcal{A}_L \cup \mathcal{A}_R \to \mathsf{dom}(\kappa_L) \cup \mathsf{dom}(\kappa_R)$. Our aim is to keep extended configurations in sync with composite configurations (i.e. those using $\oslash$), and for this reason it will be useful to extend our expression syntax with an "external return" construct:

$$e \ ::= \ \cdots \mid \mathsf{ret}_L(e) \mid \mathsf{ret}_R(e)$$

We next define the semantics for extended configurations ($\mapsto$).

$$
\begin{aligned}
(\vec{\mathcal{A}},\vec{\kappa},\phi,e) &\mapsto (\vec{\mathcal{A}},\vec{\kappa},\phi,e') \ \text{ if } e \to e' \\
(\vec{\mathcal{A}},\vec{\kappa},\phi,E[\mathfrak{o}v]) &\mapsto \begin{cases} (\vec{\mathcal{A}}\uplus_R\vec{\mathfrak{o}},\vec{\kappa}\uplus_L[\vec{\mathfrak{p}}\mapsto\vec{v}],\phi\uplus[\vec{\mathfrak{o}}\mapsto\vec{\mathfrak{p}}],E[\mathsf{ret}_L(e)]) & \text{if } \mathfrak{o}\in\mathcal{A}_L, \kappa_L(\phi(\mathfrak{o}))\,D[\vec{\mathfrak{o}}]\succ e \\ (\vec{\mathcal{A}}\uplus_L\vec{\mathfrak{o}},\vec{\kappa}\uplus_R[\vec{\mathfrak{p}}\mapsto\vec{v}],\phi\uplus[\vec{\mathfrak{o}}\mapsto\vec{\mathfrak{p}}],E[\mathsf{ret}_R(e)]) & \text{if } \mathfrak{o}\in\mathcal{A}_R, \kappa_R(\phi(\mathfrak{o}))\,D[\vec{\mathfrak{o}}]\succ e \end{cases} \\
(\vec{\mathcal{A}},\vec{\kappa},\phi,E[\mathsf{ret}_L(v)]) &\mapsto (\vec{\mathcal{A}}\uplus_L\vec{\mathfrak{o}},\vec{\kappa}\uplus_R[\vec{\mathfrak{p}}\mapsto\vec{v}],\phi\uplus[\vec{\mathfrak{o}}\mapsto\vec{\mathfrak{p}}],E[D[\vec{\mathfrak{o}}]]) \\
(\vec{\mathcal{A}},\vec{\kappa},\phi,E[\mathsf{ret}_R(v)]) &\mapsto (\vec{\mathcal{A}}\uplus_R\vec{\mathfrak{o}},\vec{\kappa}\uplus_L[\vec{\mathfrak{p}}\mapsto\vec{v}],\phi\uplus[\vec{\mathfrak{o}}\mapsto\vec{\mathfrak{p}}],E[D[\vec{\mathfrak{o}}]])
\end{aligned}
$$

where, in the last four rules, $(D,\vec{v})\in\mathsf{ulpatt}(v)$.

We continue by defining the ***internal composition*** of compatible configurations $\mathcal{C}_L \asymp_\phi \mathcal{C}_R$. We define the internal composition $\mathcal{C}_L \curlywedge_\phi \mathcal{C}_R$ to be a configuration in our new composite semantics by pattern matching on the configuration polarity and evaluation stacks according to the following rules.

Initial Configuration:

$$
\begin{aligned}
\mathcal{C}_L &= \langle\emptyset\,;\cdot\,;\cdot\,;\cdot\,;e\,;\cdot\,;\varepsilon\rangle \\
\mathcal{C}_R &= \langle\emptyset\,;\cdot\,;(E[\cdot],\varepsilon)\,;\cdot\,;\cdot\,;\varepsilon\rangle \\
\mathcal{C}_L \curlywedge \mathcal{C}_R &= (\emptyset,\emptyset,\cdot,\cdot,\cdot,E[e])
\end{aligned}
$$

Interim Configuration (case PO):

$$
\begin{aligned}
\mathcal{C}_L &= \langle\mathcal{A}_L\,;\kappa_L\,;K_L\,;t\,;e\,;\vec{\mathfrak{p}}_L,V_L\,;\vec{\mathfrak{o}}_L\rangle \\
\mathcal{C}_R &= \langle\mathcal{A}_R\,;\kappa_R\,;K_R\,;t\,;V_R\,;\vec{\mathfrak{p}}_R\rangle \\
\mathcal{C}_L \curlywedge_\phi \mathcal{C}_R &= (\mathcal{A}_L,\mathcal{A}_R,\kappa_L,\kappa_R,\phi,(K_L \curlywedge_R K_R)[e])
\end{aligned}
$$

Interim Configuration (case OP):

$$
\begin{aligned}
\mathcal{C}_L &= \langle\mathcal{A}_L\,;\kappa_L\,;K_L\,;t\,;V_L\,;\vec{\mathfrak{p}}_L\rangle \\
\mathcal{C}_R &= \langle\mathcal{A}_R\,;\kappa_R\,;K_R\,;t\,;e\,;\vec{\mathfrak{p}}_R,V_R\,;\vec{\mathfrak{o}}_R\rangle \\
\mathcal{C}_L \curlywedge_\phi \mathcal{C}_R &= (\mathcal{A}_L,\mathcal{A}_R,\kappa_L,\kappa_R,\phi,(K_L \curlywedge_L K_R)[e])
\end{aligned}
$$

where $K_L \curlywedge_{L/R} K_R$ is a single evaluation context resulting from the composition of compatible stacks, which we define as follows:

$$
\begin{aligned}
\cdot \curlywedge_{L/R} \cdot &= [\cdot] \\
((E[\cdot],\vec{\mathfrak{o}}),K_L) \curlywedge_L K_R &= (K_L \curlywedge_R K_R)[E[\mathsf{ret}_L([\cdot])]] \\
K_L \curlywedge_R ((E[\cdot],\vec{\mathfrak{o}}),K_R) &= (K_L \curlywedge_L K_R)[E[\mathsf{ret}_R([\cdot])]]
\end{aligned}
$$

Notice that there is only one case for initial configurations, and that is because the game must start from an opponent-proponent configuration where stacks are empty.

3) Bisimilarity of Semantic and Internal Composition

We begin by defining (weak) bisimilarity for the semantic and internal composition. A set $\mathcal{R}$ with elements of the form (X_1,X_2), where X_1 is a pair of the form $\mathcal{C}_L \oslash_\phi \mathcal{C}_R$ and X_2 is from the composite semantics, is an *si-bisimulation* if for all $(X_1,X_2)\in R$:

- if $X_1 \to' X_1'$ then $X_2 \mapsto^* X_2'$ and $(X_1',X_2')\in\mathcal{R}$;
- if $X_2 \mapsto X_2'$ then $X_1 \to'^* X_1'$ and $(X_1',X_2')\in\mathcal{R}$.

We say that X_1,X_2 are *si-bisimilar*, and write $X_1 \sim_{si} X_2$, if there is an si-bisimulation $\mathcal{R}$ such that $X_1\mathcal{R}X_2$.

Lemma 28. *Given configurations $\mathcal{C}_L \asymp_\phi \mathcal{C}_R$, it is the case that $(\mathcal{C}_L \oslash_\phi \mathcal{C}_R) \sim_{si} (\mathcal{C}_L \curlywedge_\phi \mathcal{C}_R)$.*

4) Soundness

To prove soundness of the game-LTS semantics, we want to show that syntactic composition can be obtained from semantic counterpart and vice versa. We have si-bisimilarity between semantic and internal composition, we only need to show that internal composition is related to syntactic composition under some notion of equivalence.

Lemma 29. *Let $\vdash e : T$ by any expression and E any evaluation context such that $\vdash E[e] : \mathsf{unit}$, and let*

$$\mathcal{C}_L = \langle \emptyset\,;\cdot\,;\cdot\,;\cdot\,;e\,;\cdot\,;\varepsilon\rangle \quad and \quad \mathcal{C}_R = \langle \emptyset\,;\cdot\,;(E[\cdot]_T,\varepsilon)\,;\cdot\,;\cdot\,;\varepsilon\rangle.$$

Then, $E[e]\Downarrow$ iff there is a complete play t and some ϕ such that $t \in CP(\mathcal{C}_L)$ and $\bar{t}^\phi\ \mathsf{ret}(()) \in CP^+(\mathcal{C}_R)$.

Proof. Suppose $E[e]\Downarrow$, and recall that $\mathcal{C}_L \curlywedge \mathcal{C}_R = (\emptyset,\emptyset,\cdot,\cdot,\cdot,E[\mathsf{ret}_R(e)])$. Then:

1) By inspection of the composite semantics, we have that $\mathcal{C}_L \curlywedge \mathcal{C}_R$ reaches $()$.
2) By si-bisimilarity (Lemma 28) we have that $\mathcal{C}_L \oslash \mathcal{C}_R$ reaches $()$, say with final name bijection π.
3) By definition of semantic composition, we know there are traces $t \in CP(\mathcal{C}_L)$ and $t'\ \mathsf{ret}(()) \in CP^+(\mathcal{C}_R)$ such that $t' = \bar{t}^\phi$.

Conversely, suppose there is complete $t \in CP(\mathcal{C}_L)$ such that $\bar{t}^\phi\ \mathsf{ret}(()) \in CP^+(\mathcal{C}_R)$. Then:

1) By definition of semantic composition we have that $\mathcal{C}_L \oslash \mathcal{C}_R$ reaches $()$, with final bijection ϕ.
2) By si-bisimilarity (Lemma 28) we have that $\mathcal{C}_L \curlywedge \mathcal{C}_R$ reaches $()$.
3) By inspection of the composite semantics, we obtain that $E[e]$ reaches $()$.

□

Corollary 30 (Complete-play Soundness)**.** *Given $\vdash e_1, e_2 : T$, if $CP(\mathcal{C}_{e_1}) = CP(\mathcal{C}_{e_2})$ then $e_1 \equiv e_2$.*

Proof. Suppose there is some evaluation context such that $E[e_1]\Downarrow$ and $E[e_2]\Uparrow$. Then, by previous Lemma, there is some $t \in CP(\mathcal{C}_{e_1})$ and $\bar{t}^\phi \in CP(\mathcal{C}_E)$. By the same Lemma, and the fact that $E[e_2]\Uparrow$, we have that $t \notin CP(\mathcal{C}_{e_2})$. □

Proposition 31 (Soundness)**.** *Given $\vdash e_1, e_2 : T$, if $OV_{tl}(\mathcal{C}_{e_1}) = OV_{tl}(\mathcal{C}_{e_2})$ then $e_1 \equiv e_2$.*

Proof. Suppose there is some applicative context E such that $E[e_1]\Downarrow$ and $E[e_2]\Uparrow$. Then, by Lemma 29, there is some $t \in CP(\mathcal{C}_{e_1}) \setminus CP(\mathcal{C}_{e_2})$ and ϕ such that $\bar{t}^\phi\ \mathsf{ret}(()) \in CP^+(\mathcal{C}_E)$. Since E is applicative, t must be top-linear. Taking $S = \{\pi \cdot \llcorner t' \lrcorner \mid t' \sqsubseteq t\}$, we have $S \in OV_{tl}(\mathcal{C}_{e_1})$. Let us suppose that $S \in OV_{tl}(\mathcal{C}_{e_2})$. Then, there is $\hat{t} \in CP(\mathcal{C}_{e_2})$ with $\{\pi \cdot \llcorner t' \lrcorner \mid t' \sqsubseteq \hat{t}\} = S$. Then, by Lemma 26(7) we have that $\bar{\hat{t}}^\phi \in \mathsf{Pls}(\mathcal{C}_E)$. Now observe that, since $t, \hat{t}$ are complete:

$$\bar{t}^\phi = o_0 p_1 \ldots o_1 \cdots p_n \ldots o_n \text{ and } \bar{\hat{t}}^\phi = \hat{o}_0 \hat{p}_1 \ldots \hat{o}_1 \cdots \hat{p}_{\hat{n}} \ldots \hat{o}_{\hat{n}}$$

where, for each $i > 0$, o_i returns p_i and $\hat{o}_i$ returns $\hat{p}_i$. By hypothesis, we have that $o_0 = \hat{o}_0$ and thus P-innocence implies that $p_1 \sim \hat{p}_1$. WLOG we can assume that $p_1 = \hat{p}_1$(since otherwise we can consider permutation variants of $t, \hat{t}$ which agree on the corresponding fresh names). Again by hypothesis there is a prefix t' of $\bar{t}^\phi$ such that $\ulcorner t' \urcorner = \hat{o}_0 \hat{p}_1 \hat{o}_1 = o_0 p_1 \hat{o}_1$. By O-innocence (on S) we have $o_1 \sim \hat{o}_1$ and, WLOG, $o_1 = \hat{o}_1$. Applying the same reasoning repeatedly we obtain that $\ulcorner \bar{t}^\phi \urcorner = \ulcorner \bar{\hat{t}}^\phi \urcorner$, and therefore $\bar{\hat{t}}^\phi\ \mathsf{ret}(()) \in CP^+(\mathcal{C}_E)$. Hence, by Lemma 29 we have that $E[e_2]\Downarrow$, a contradiction. Thus, $S \notin OV_{tl}(\mathcal{C}_{e_2})$. □

B. Definability and Completeness for Game-LTS

We prove completeness via a definability argument which follows the lines of game-semantics definability proofs (e.g. [12], [40]). We show that, for each complete play t, the set $OV(t)$ has a matching evaluation context E such that $\mathcal{C}_E$ realises (the *duals* of) all plays in $OV(t)$.

For the definability argument it will be useful to consider open terms, where the open variables are instantiated with (possibly abstract) values. We therefore redefine initial configurations to be of the form $\langle \Delta \vdash e : T\rangle$ and extend the game-LTS of Fig. 3 with initialisation moves $(\underline{\mathsf{call}}(?, D[\vec{\mathfrak{o}}]))$ and the initialisation rule:

$$\langle \Delta \vdash e : T\rangle \xrightarrow{\underline{\mathsf{call}}(?,D[\vec{\mathfrak{o}}])} \langle \vec{\mathfrak{o}}\,;\cdot\,;\cdot\,;\cdot\,;e[\vec{\mathfrak{o}}/\vec{x}]\,;\varepsilon\,;\vec{\mathfrak{o}}\rangle \qquad \text{if } (D,\vec{\mathfrak{o}}) \in \mathsf{ulpatt}(T_1 * \cdots * T_n) \tag{INIT}$$

assuming $\Delta = \{x_1 : T_1 \ldots, x_n : T_n\}$. Accordingly, *extended complete traces* are given by the grammar:

$$ECT \;\rightarrow\; \underline{\mathsf{call}}(?, D[\vec{\mathfrak{o}}])\, CT_P$$

whereas *extended traces* are prefixes thereof. The notions of O-view and P-view are defined as in plain traces, with the caveat that the second move in an extended trace is justified by the first move (if they exist). Hence, *extended plays* are legal extended traces satisfying the conditions of Definition 20.

We denote configurations in the extended LTS by $\mathcal{K}$ and variants for clarity. We write $CEP(\Delta \vdash e : T)$ for the set of complete extended plays produced by the extended LTS starting from $\langle \Delta \vdash e : T\rangle$. We then let

$$EPV(\Delta \vdash e : T) = \{\ulcorner t' \urcorner \mid t' \sqsubseteq t \in CEP(\Delta \vdash e : T) \wedge |t'|\ \text{even}\}.$$

Note that we require that only even-length P-views are in EPV (this is for technical convenience).

Definition 32. We call a set of extended plays $\mathcal{F}$ a *viewfunction* if:

1) for all $t \in \mathcal{F}$, $\ulcorner t \urcorner = t$, and $\mathcal{F}$ is even-prefix closed and also closed under name permutations (i.e. from $Perm$);
2) if $tm_1, tm_2 \in \mathcal{F}$ then $tm_1 \sim tm_2$;
3) if $m_1 t_1, m_2 t_2 \in \mathcal{F}$ then $m_1 \sim m_2$.

Given types $\vec{T}, T$, we write $\vec{T} \vdash \mathcal{F} : T$ if either $\mathcal{F}$ is empty or it contains $\underline{\mathsf{call}}(?, D_0[\vec{\mathfrak{o}}_0])\, \mathsf{ret}(D_1[\vec{\mathfrak{p}}_1])$ such that $(D_0, \vec{\mathfrak{o}}_0) \in \mathsf{ulpatt}\, T_1 * \cdots * T_n$ and $D_1[\vec{\mathfrak{p}}] : T$.

Finally, we call $\mathcal{F}$ *finite-orbit* if its set of orbits (under permutations):

$$\mathcal{O}(\mathcal{F}) = \{[t] \mid t \in \mathcal{F}\}$$

is finite. We then set $\|\mathcal{F}\| = |\mathcal{O}(\mathcal{F})|$. $\square$

Thus, a viewfunction is a set of extended P-views representing a deterministic proponent behaviour. Condition 3 in the definition above imposes such a notion of determinacy (P-*innocence*): each next opponent move is determined by the preceding O-view. Condition 4 imposes uniqueness of the initialising move and, combined with P-innocence, ensures that typings are unique for non-empty $\mathcal{F}$.

Proposition 33 (Definability)**.** *For any finite-orbit $\vec{T} \vdash \mathcal{F} : T$ there is a term $\Delta \vdash e : T$ evaluation context E such that $EPV(\Delta \vdash e : T) = \mathcal{F}$.*

Proof. Suppose that $\Delta = \{x_1 : T_1, \ldots, x_n : T_n\}$. We do induction on $\|\mathcal{F}\|$. For the base case ($\mathcal{F} = \emptyset$) we can set $E \stackrel{\text{def}}{=} \bot_T$. Suppose now $\|\mathcal{F}\| > 0$, so there is some $\underline{\mathsf{call}}(?, D_0[\vec{\mathfrak{o}}_0])\, m_0 \in \mathcal{F}$. If there is some evaluation term $\Delta \vdash e' : T$ such that

$$\mathcal{F} = \{mt \in EPV(\Delta \vdash e' : T) \mid m \sim \underline{\mathsf{call}}(?, D_0[\vec{\mathfrak{o}}_0])\} \tag{$*$}$$

then we can set $e \stackrel{\text{def}}{=} \mathsf{if}\ \vec{x} = D_0\ \mathsf{then}\ e'\ \mathsf{else}\ \bot_T$, where $\vec{x} = D_0$ is a (definable) macro that checks whether the ultimate pattern of $\vec{x}$ is D_0.

We next produce e'. The move m_0 can either be a return or an application of some $\mathfrak{o}_{0i}$. Suppose first that $m_0 = \mathsf{ret}(D_1[\vec{\mathfrak{p}}])$, and let $\vec{\mathfrak{p}} = \mathfrak{p}_1 \cdots \mathfrak{p}_k$. For each j, assuming $\mathfrak{p}_j$ has function type $T_j = T'_j \to T''_j$, consider the set:

$$\mathcal{F}_j = \bigcup \{[\, \underline{\mathsf{call}}(?, (D_0, D)[\vec{\mathfrak{o}}_0, \vec{\mathfrak{o}}])\, t\,] \mid \underline{\mathsf{call}}(?, D_0[\vec{\mathfrak{o}}_0])\, \mathsf{ret}(D_1[\vec{\mathfrak{p}}])\, \underline{\mathsf{call}}(\mathfrak{p}_j, D[\vec{\mathfrak{o}}])\, t \in \mathcal{F}\}$$

We can see that $\mathcal{F}_j$ is a viewfunction and, moreover, $\|\mathcal{F}_j\| < \|\mathcal{F}\|$, so by IH we obtain a term $\Delta, y : T'_j \vdash e_j : T''_j$ such that $EPV(e_j) = \mathcal{F}_j$. Thus, taking $e' \stackrel{\text{def}}{=} D_1[\overrightarrow{\lambda y.e_j}]$ we can satisfy ($*$).

Finally, suppose that $m_0 = \mathsf{call}(\mathfrak{o}_{0i}, D_1[\vec{\mathfrak{p}}])$, with $\vec{\mathfrak{p}} = \mathfrak{p}_1 \cdots \mathfrak{p}_k$, and assume that $\mathfrak{p}_j : T'_j \to T''_j$ for each $j = 1, \ldots, k$, while $\mathfrak{o}_{0i} : T'_0 \to T''_0$. In this case, proponent responds by calling one of the opponent functions ($\mathfrak{o}_{0i}$). We consider the following sets of P-views which determine proponent's behaviour after each subsequent opponent move:

$$\mathcal{F}_R = \bigcup \{[\, \underline{\mathsf{call}}(?, (D_0, D)[\vec{\mathfrak{p}}_0, \vec{\mathfrak{p}}])\, t\,] \mid \underline{\mathsf{call}}(?, D_0[\vec{\mathfrak{o}}_0])\, \mathsf{call}(\mathfrak{o}_{0i}, D_1[\vec{\mathfrak{p}}])\, \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])\, t \in \mathcal{F}\}$$
$$\mathcal{F}_j = \bigcup \{[\, \underline{\mathsf{call}}(?, (D_0, D)[\vec{\mathfrak{p}}_0, \vec{\mathfrak{p}}])\, t\,] \mid \underline{\mathsf{call}}(?, D_0[\vec{\mathfrak{p}}_0])\, \mathsf{call}(\mathfrak{o}_{0i}, D_1[\vec{\mathfrak{p}}])\, \underline{\mathsf{call}}(\mathfrak{p}_j, D[\vec{\mathfrak{o}}])\, t \in \mathcal{F}\}$$

we can see that each of the above has smaller measure than $\mathcal{F}$ hence, by applying the IH, we obtain terms $\Delta, y : T''_0 \vdash e_R : T$ and $\Delta, z : T'_j \vdash e_j : T''_j$ (for each j). Then, taking

$$e' \stackrel{\text{def}}{=} \mathsf{let}\ y = \mathsf{proj}_i(D_0)(x)(D_1[\overrightarrow{\lambda z.e_j}])\ \mathsf{in}\ e_R$$

we can satisfy ($*$). Here, $\mathsf{proj}_i(D_0)(x)$ is a macro obtaining the i-th function component from an x with ultimate pattern D_0. This completes the proof. $\square$

The remainder of this section follows closely [7].

Definition 34. Suppose Σ, Σ' are sets of P-starting O-views. We let $\Sigma \leq \Sigma'$ if for all $\mathcal{F} \in \Sigma$ there is $\mathcal{F}' \in \Sigma'$ such that $\mathcal{F}' \subseteq \mathcal{F}$. $\square$

Let us fix a bijection $\phi : ONames \stackrel{\cong}{\to} PNames$. For each set of plays S, we shall write $\overline{S}$ for the set $\{\overline{t}^{\phi} \mid t \in S\}$.

Lemma 35. *Given $\vdash e_1, e_2 : T$, if $e_1 \equiv e_2$ then $OV(\mathcal{C}_{e_1}) \leq OV(\mathcal{C}_{e_2}) \leq OV(\mathcal{C}_{e_1})$.*

Proof. By symmetry, it suffices to show one inclusion. Suppose $e_1 \equiv e_2$ and let $t \in CP(\mathcal{C}_{e_1})$ be a top-linear complete play. Consider the set of extended plays:

$$\mathcal{F} = \{\underline{\mathsf{call}}(?, D_0[\phi(\vec{\mathfrak{p}})])\, \overline{t'}^{\phi} \mid \mathsf{ret}(D_0[\vec{\mathfrak{p}}])\, t' \in OV(t) \wedge |t'|\ \text{odd}\} \cup \{\underline{\mathsf{call}}(?, D_0[\phi(\vec{\mathfrak{p}})])\, \overline{t'}^{\phi}\, \mathsf{ret}(()) \mid \ulcorner t \urcorner = \mathsf{ret}(D_0[\vec{\mathfrak{p}}])\, t'\}$$

$$\frac{(D,\vec{v}) \in \mathsf{ulpatt}(v) \qquad t' = t + \mathsf{ret}(D[\vec{\mathfrak{p}}]) \qquad \kappa' = \kappa \uplus [\vec{\mathfrak{p}} \mapsto \vec{v}^{\vec{\mathfrak{o}}}]}{\langle A\,;\kappa\,;\cdot\,;t\,;v\,;\vec{\mathfrak{p}}'\,;\vec{\mathfrak{o}}\rangle \xrightarrow{\mathsf{ret}(D[\vec{\mathfrak{p}}])} \langle \emptyset\,;\kappa'\,;\cdot\,;t'\,;\cdot\,;\vec{\mathfrak{p}}\rangle}\ \textsc{PropRetBarb}$$

$$\frac{\mathfrak{p}_i : T_1 \to T_2 \qquad \kappa(\mathfrak{p}_i) = v^{\vec{\mathfrak{o}}'} \qquad (D,\vec{\mathfrak{o}}) \in \mathsf{ulpatt}(T_1) \qquad t' = t + \underline{\mathsf{call}}(\mathfrak{p}_i, D[\vec{\mathfrak{o}}])}{\langle A\,;\kappa\,;\cdot\,;t\,;\cdot\,;\vec{\mathfrak{p}}\rangle \xrightarrow{\underline{\mathsf{call}}(\mathfrak{p}_i,D[\vec{\mathfrak{o}}])} \langle A \uplus \vec{\mathfrak{o}}^{\varepsilon}\,;\kappa\,;\cdot\,;t'\,;e\,;\cdot\,;\vec{\mathfrak{o}}'\rangle}\ \textsc{OpAppBarb}$$

$$\frac{(D,\vec{v}) \in \mathsf{ulpatt}(v) \qquad K \neq \cdot \qquad t' = t + \mathsf{ret}(D[\vec{\mathfrak{p}}]) \qquad \kappa' = \kappa \uplus [\vec{\mathfrak{p}} \mapsto \vec{v}^{\vec{\mathfrak{o}}}]}{\langle A\,;\kappa\,;K\,;t\,;v\,;\vec{\mathfrak{p}}', V\,;\vec{\mathfrak{o}}\rangle \xrightarrow{\mathsf{ret}(D[\vec{\mathfrak{p}}])} \langle A\,;\kappa'\,;K\,;t'\,;V\,;\vec{\mathfrak{p}}', \vec{\mathfrak{p}}\rangle}\ \textsc{PropRet}$$

$$\frac{[\underline{\mathsf{call}}(\mathfrak{p}_i, D[\vec{\mathfrak{o}}])] \subseteq \mathsf{next}_O(t) \qquad \mathfrak{p}_i : T_1 \to T_2 \qquad \kappa(\mathfrak{p}_i) = v^{\vec{\mathfrak{o}}'} \qquad (D,\vec{\mathfrak{o}}) \in \mathsf{ulpatt}(T_1) \qquad t' = t + \underline{\mathsf{call}}(\mathfrak{p}_i, D[\vec{\mathfrak{o}}]) \qquad K \neq \cdot}{\langle A\,;\kappa\,;K\,;t\,;V\,;\vec{\mathfrak{p}}\rangle \xrightarrow{\underline{\mathsf{call}}(\mathfrak{p}_i,D[\vec{\mathfrak{o}}])} \langle A \uplus \vec{\mathfrak{o}}^{\vec{\mathfrak{p}}}\,;\kappa\,;K\,;t'\,;e\,;\vec{\mathfrak{p}}, V\,;\vec{\mathfrak{o}}', \vec{\mathfrak{o}}\rangle}\ \textsc{OpApp}$$

Fig. 4. The Game Labelled Transition System, top-linear version (PropTau,PropApp,OpRet rules as in Figure 3).

By determinacy of the LTS (for both O and P), we have that $\mathcal{F}$ is a viewfunction. Hence, by Proposition 33, there is a term $x : T \vdash e : \mathsf{unit}$ such that $\mathcal{F} = EPV(e)$. Now, define the context:

$$E \stackrel{\text{def}}{=} \mathsf{let}\, x = [\cdot]_T \,\mathsf{in}\, e$$

By inspection of the LTS rules, we can see that the plays in $CP(\mathcal{C}_E)$ and the extended plays in $CEP(e)$ are the same, modulo adjusting the first move of extended plays (from an initialisation move to an O-return one) and removing the last one ($\mathsf{ret}(())$). Hence, $PV(\mathcal{C}_E) = \overline{OV(t)}$. Moreover, $\mathcal{C}_E$ produces all traces in $PV(\bar{t}^{\phi})$, and therefore $\bar{t}^{\phi} \in CP(\mathcal{C}_E)$. By simulating the reduction producing $\bar{t}^{\phi}$ in the extended LTS (for e), we can see that $\bar{t}^{\phi}\,\mathsf{ret}(()) \in CP^{+}(\mathcal{C}_E)$. Hence, by Lemma 29, $E[e_1] \Downarrow$ and thus, by hypothesis, $E[e_2] \Downarrow$. Again, by Lemma 29 and using the fact that the LTS is closed with respect to name permutations, there is a play $s \in CP(\mathcal{C}_{e_2})$ such that $\bar{s}^{\phi}\,\mathsf{ret}(()) \in CP^{+}(\mathcal{C}_E)$. By $\bar{s}^{\phi} \in CP(\mathcal{C}_E)$, we have that $PV(\bar{s}^{\phi}) \subseteq PV(\mathcal{C}_E) = \overline{OV(t)}$ and, hence, $OV(s) \subseteq OV(t)$. Thus, $OV(\mathcal{C}_{e_1}) \le OV(\mathcal{C}_{e_2})$. ☐

Lemma 36. *If $OV(\mathcal{C}_{e_1}) \le OV(\mathcal{C}_{e_2}) \le OV(\mathcal{C}_{e_1})$ then $OV(\mathcal{C}_{e_1}) = OV(\mathcal{C}_{e_2})$.*

Proof. Let $S_1 \in OV(\mathcal{C}_{e_1})$. By hypothesis, there are $T \in OV(\mathcal{C}_{e_2})$ and $S_2 \in OV(\mathcal{C}_{e_1})$ such that $S_2 \subseteq T \subseteq S_1$. Let $t_1, t_2 \in CP(\mathcal{C}_{e_1})$ be such that $S_i = OV(t_i)$. Suppose that $S_1 \neq S_2$, so $t_1 \neq t_2$. As the LTS is P-deterministic, there is an odd-length play t and $o_1 \not\sim o_2$ such that $to_1, to_2 \in \mathsf{Pls}(\mathcal{C}_{e_1})$ and $\llcorner t \lrcorner o_i \in OV(t_i)$. In particular, $\llcorner t \lrcorner o_2 \in S_2 \subseteq S_1$, so $\llcorner t \lrcorner o_2 \in S_1$. But then $\llcorner t \lrcorner o_1, \llcorner t \lrcorner o_2 \in S_1$, contradicting O-determinacy. Thus, $S_1 = S_2 = T$ and, hence, $S_1 \in OV(\mathcal{C}_{e_2})$. ☐

Proposition 37 (Completeness)**.** *Given $\vdash e_1, e_2 : T$, if $e_1 \equiv e_2$ then $OV_{tl}(\mathcal{C}_{e_1}) = OV_{tl}(\mathcal{C}_{e_2})$.*

Proof. It suffices to show one inclusion. Suppose $e_1 \equiv e_2$ so by the previous two lemmata, $OV(\mathcal{C}_{e_1}) = OV(\mathcal{C}_{e_2})$. Let $t \in CP(\mathcal{C}_{e_1})$ be top linear, so $OV(t) \in OV(\mathcal{C}_{e_2})$, say $OV(t) = OV(s)$ for some $s \in CP(\mathcal{C}_{e_2})$. As all the plays in $OV(s)$ are top-linear, so is s. Hence, $OV(t) \in OV_{tl}(\mathcal{C}_{e_2})$. ☐

C. Correspondence of LTS and game-LTS

We next show that our (plain) LTS and game-LTS produce the same notion of term equivalence. In this section, by game-LTS we refer to the LTS of Figure 4, so in particular all plays examined will be top-linear (cf. Lemma 38). Moreover, by plain LTS we intend the LTS of Figure 2 albeit with the modification that in rules PropRet, PropApp, OpRet, OpApp the transition is labelled with the corresponding move (i.e. the one that ends up in the trace stored in the target configuration) and not with τ. In particular, only transitions triggered by PropTau are labelled with τ. We can see that this modification does not essentially alter the LTS, and in particular it produces the same M-components.

Lemma 38. *Let $CP'(\mathcal{C})$ be the complete plays produced from $\mathcal{C}$ using the rules of the LTS in Figure 4. Then, $OV_{tl}(\mathcal{C}) = \{OV(t) \mid t \in CP'(\mathcal{C})\}$.*

We will show that there is a translation from one LTS to the other that preserves traces and the functions M.

Given components $\mathcal{A}, \kappa, t$ from a reachable configuration (from the game-LTS), we shall define corresponding plain-LTS components and a function ψ from $ONames$ to abstract function names:[3]

$$(A, M, \hat{t}, \hat{s}, \psi) \in (\mathcal{A}, \kappa, t)^{\circ}.$$

[3]Note that, in this section, all traces are P-starting.

In particular, for each $\mathfrak{o} \in \mathsf{dom}(\psi)$, $\psi(\mathfrak{o}) = \alpha^i$, for some α, i. We may write

$$\psi(\mathfrak{o}) \doteq \alpha \text{ if } \psi(\mathfrak{o}) = \alpha^i \text{ for some } i.$$

We call a triple $(\mathcal{A}, \kappa, t)$ *compatible* if

$$ONames(\kappa) \subseteq \mathsf{dom}(\mathcal{A}) = ONames(t) \;\wedge\; \mathsf{dom}(\kappa) = PNames(t)$$

where $ONames(X)$ are the O-names featuring in X, and similarly for $PNames(X)$.

First, given any incomplete top-linear play t with $|t| > 1$, we can split t as:

$$t = t_{cp}\, o\, t_{lo}$$

where t_{cp} a complete trace, o is a top-level O-call and t_{lo} contains no top-level moves. Then, we let $\llcorner t \lrcorner_\circ$ be the suffix t' of $\llcorner t \lrcorner$ satisfying the condition:

$$\llcorner t \lrcorner \;=\; \llcorner t_{cp}\, o \lrcorner \cdots t'$$

such that t' contains exactly one P-call, which is at its start. If t is complete or t_{lo} is empty, then $\llcorner t \lrcorner_\circ = \cdot$.

Lemma 39. *Let $\mathcal{C}$ be an initial configuration and suppose that $\mathcal{C} \xrightarrow{t}\!\!\!\!\rightarrow \mathcal{C}'$ in the LTS of Figure 4. Then:*

1) *t is a play and if $\mathcal{C}'$ has components $\mathcal{A}, \kappa$ then the names in t are precisely $\mathcal{A} \cup \mathsf{dom}(\kappa)$;*
2) *for any name permutation π, $\mathcal{C} \xrightarrow{\pi\cdot t}\!\!\!\!\rightarrow \pi \cdot \mathcal{C}'$;*
3) *if $\mathcal{C}' = \langle \_\,; \_\,; \_\,; t\,; \_\,; \vec{\mathfrak{p}} \rangle$ then the P-names in $\llcorner t \lrcorner_\circ$ are $\vec{\mathfrak{p}}$ (in the same order).*

We shall also be using the following lemma regarding name permutations (cf. [40]).

Lemma 40. *Given sequences of moves $t_1 \sim t_2$ and names x_1, x_2 of the same kind (either both O-names or both P-names), if x_i is fresh for t_i ($i = 1, 2$) then $t_1 x_1 \sim t_2 x_2$.*

Definition 41. Given compatible $(\mathcal{A}, \kappa, t)$, we define $(\mathcal{A}, \kappa, t)^\circ$ by induction on $|t|$. For the base case:

$$(\mathcal{A}, \kappa, \cdot)^\circ = \{(\emptyset, \cdot, \cdot, \cdot, \cdot)\}$$

If $t = t' + m$ then, for each $(A', M', \hat{t}', \hat{s}', \psi') \in (\mathcal{A}', \kappa', t')^\circ$ where $\mathcal{A}', \kappa'$ the restrictions of $\mathcal{A}, \kappa$ respectively to the names in t', we include in $(\mathcal{A}, \kappa, t)^\circ$ a triple $(A, M, \hat{t}, \hat{s}, \psi)$ defined by case analysis on m:

- If $m = \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])$ then
  - $M = M'[\hat{t}' + \underline{\mathsf{ret}}(D[\vec{\alpha}])]$ where: if M is defined in $\hat{t}'$ then we require that $\mathsf{next}_M(\hat{t}') = \underline{\mathsf{ret}}(D[\vec{\alpha}])$ (for some $\vec{\alpha}$), otherwise $\vec{\alpha}$ are fresh;
  - $A = A' \uplus \vec{\alpha}^{j,\vec{v}}$, for the least j such that $\vec{\alpha}^{j,\cdots} \notin A'$, and $\vec{v} = \psi'(\kappa(\vec{\mathfrak{p}}))$ with $\vec{\mathfrak{o}}^{\vec{\mathfrak{p}}} \in \mathcal{A}$;
  - $\hat{s} = \hat{s}'$ and $\psi = \psi'[\vec{\mathfrak{o}} \mapsto \vec{\alpha}^j]$;
  - if $t'' \sqsubseteq t$ ends in the last open call in t (which must be an O-move) and $(\mathcal{A}'', \kappa'', t'')^\circ = (A'', M'', \hat{t}'', \psi'')$ then $\hat{t} = \hat{t}''$.
- If $m = \underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$ with $\mathfrak{p}$ the i-th P-name in $\llcorner t \lrcorner_\circ$, then
  - if t' is complete then $M = M'$, otherwise $M = M'[\hat{t}' + \underline{\mathsf{call}}(i, D[\vec{\alpha}])]$, where: if M is defined in $\hat{t}'$ then we require that $\mathsf{next}_M(\hat{t}') = \underline{\mathsf{call}}(i, D[\vec{\alpha}])$ (for some $\vec{\alpha}$), otherwise $\vec{\alpha}$ are fresh;
  - $A = A' \uplus \vec{\alpha}^{j,\vec{v}}$, for the least j such that $\vec{\alpha}^{j,\cdots} \notin A'$, and $\vec{v} = \psi'(\kappa(\vec{\mathfrak{p}}))$ with $\vec{\mathfrak{o}}^{\vec{\mathfrak{p}}} \in \mathcal{A}$;
  - if t' is complete then $\hat{s} = \hat{s}' + \underline{\mathsf{call}}(i, D[\vec{\alpha}])$, otherwise $\hat{s} = \hat{s}'$;
  - $\psi = \psi'[\vec{\mathfrak{o}} \mapsto \vec{\alpha}^j]$;
  - if t' is complete then $\hat{t} = \cdot$, otherwise $\hat{t} = \hat{t}' + \underline{\mathsf{call}}(i, D[\vec{\alpha}])$.
- If $m = \mathsf{ret}(D[\vec{\mathfrak{p}}])$ then
  - if t is complete then $M = M'$, $A = A'$, $\psi = \psi'$, $\hat{s} = \hat{s}' + \mathsf{ret}(D)$ and $\hat{t} = \cdot$;
  - otherwise, $M = M'[\hat{t}' + \mathsf{ret}(D)]$, $A = A'$, $\psi = \psi'$, $\hat{s} = \hat{s}'$ and $\hat{t} = \hat{t}' + \mathsf{ret}(D)$.
- If $m = \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])$ then, assuming $\psi'(\mathfrak{o}) \doteq \alpha$,
  - $M = M'[\mathsf{call}(\alpha, D)]$, $A = A'$, $\psi = \psi'$, $\hat{t} = \mathsf{call}(\alpha, D)$ and $\hat{s} = \hat{s}'$.

Given a play t we then define:

$$(t)^\circ = \{(M, \hat{t}, \hat{s}, \psi) \mid \exists \mathcal{A}, \kappa, A.\ (\mathcal{A}, \kappa, t) \text{ compatible} \wedge (\mathcal{A}, \kappa, t)^\circ = (A, M, \hat{t}, \hat{s}, \psi)\}.$$

□

Observe above that the components $M, \hat{t}, \hat{s}, \psi$ are defined using solely t, and thus $(t)^\circ$ is well defined. Moreover, any two elements of $(\mathcal{A}, \kappa, t)$ are equal up to permutation of O-names.

Lemma 42. *Given compatible* $(\mathcal{A},\kappa,t),(\mathcal{A}',\kappa',t)$ *and some* $(M,\hat{t},\hat{s},\psi)$*:*

1) $\exists A.(A,M,\hat{t},\hat{s},\psi)\in(\mathcal{A},\kappa,t)^\circ \iff \exists A'.(A',M,\hat{t},\hat{s},\psi)\in(\mathcal{A}',\kappa',t)^\circ$;
2) $\forall(A_1,M_1,\hat{t}_1,\hat{s}_1,\psi_1),(A_2,M_2,\hat{t}_2,\hat{s}_2,\psi_2)\in(\mathcal{A},\kappa,t)^\circ.\,\exists\pi.\,(A_1,M_1,\hat{t}_1,\hat{s}_1,\psi_1)=\pi\cdot(A_2,M_2,\hat{t}_2,\hat{s}_2,\psi_2)$, *where* π *a permutation of* O*-names.*

Given a play t (from the game-LTS) and a map ψ produced as above, we define its translated O-view, $\llcorner t\lrcorner_\psi$, by setting $\llcorner\cdot\lrcorner_\psi=\cdot$ and:

$$
\begin{aligned}
\llcorner t'\ \mathsf{ret}(D[\vec{\mathfrak{p}}])\lrcorner_\psi &= \begin{cases}\cdot & \text{if } t'\ \mathsf{ret}(D[\vec{\mathfrak{p}}]) \text{ is complete}\\ \llcorner t''\lrcorner_\psi\,\mathsf{ret}(D) & \text{otherwise, and } t''\sqsubseteq t' \text{ ends in last open call of } t'\end{cases}\\
\llcorner t'\ \mathsf{call}(\mathfrak{o},D[\vec{\mathfrak{p}}])\lrcorner_\psi &= \mathsf{call}(\alpha,D) \qquad \text{if } \psi(\mathfrak{o})\doteq\alpha\\
\llcorner t'\ \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])\lrcorner_\psi &= \llcorner t'\lrcorner_\psi\,\underline{\mathsf{ret}}(D[\vec{\alpha}]) \qquad \text{if } \psi(\vec{\mathfrak{o}})\doteq\vec{\alpha}\\
\llcorner t'\ \underline{\mathsf{call}}(\mathfrak{p},D[\vec{\mathfrak{o}}])\lrcorner_\psi &= \begin{cases}\cdot & \text{if } t' \text{ is complete}\\ \llcorner t'\lrcorner_\psi\,\underline{\mathsf{call}}(i,D[\vec{\alpha}]) & \text{otherwise, with } \psi(\vec{\mathfrak{o}})=\vec{\alpha}^{\cdots}\end{cases}
\end{aligned}
$$

where, in the latter case, $\mathfrak{p}$ should be the i-th P-name in $\llcorner t'\lrcorner_\circ$. We also define the translated top-level view $(t)_\psi^\top$ of t by $(\cdot)_\psi^\top=\cdot$ and:

$$
\begin{aligned}
(t_{cp}o\,t_{lo})_\psi^\top &= (t_{cp}\,o)_\psi^\top && \text{if } t_{lo}\neq\cdot\\
(t'\ \mathsf{ret}(D[\vec{\mathfrak{p}}]))_\psi^\top &= (t'')_\psi^\top\,\mathsf{ret}(D) && \text{if } t'\ \mathsf{ret}(D[\vec{\mathfrak{p}}]) \text{ complete, and } t''\sqsubseteq t' \text{ ends}\\
& && \text{in last open call of } t'\\
(t'\ \underline{\mathsf{call}}(\mathfrak{p},D[\vec{\mathfrak{o}}]))_\psi^\top &= (t')_\psi^\top\,\underline{\mathsf{call}}(i,D[\vec{\alpha}]) && \text{if } t' \text{ complete, with } \psi(\vec{\mathfrak{o}})\doteq\vec{\alpha}
\end{aligned}
$$

and where $\mathfrak{p}$ the i-th P-name in $\llcorner t'\lrcorner$. We can show the following.

Lemma 43. *Given* $t_1,t_2\sqsubseteq t$ *that have same-parity length, if* $\llcorner t_1\lrcorner_\psi=\llcorner t_2\lrcorner_\psi$ *then* $\llcorner t_1\lrcorner_\circ\sim\llcorner t_2\lrcorner_\circ$.

Proof. Assuming $\llcorner t_1\lrcorner_\psi=\llcorner t_2\lrcorner_\psi$, we show $\llcorner t_1\lrcorner_\circ\sim\llcorner t_2\lrcorner_\circ$ by induction on $|\llcorner t_1\lrcorner_\psi|=|\llcorner t_2\lrcorner_\psi|$. The base case (for length 0) is straightforward. Suppose now $\llcorner t_i\lrcorner_\psi=\hat{t}\,x$ $(i=1,2)$, and do case analysis on x:

- If $x=\mathsf{call}(\alpha,D)$ then $\llcorner t_i\lrcorner_\psi=\mathsf{call}(\alpha,D)$ $(i=1,2)$. Hence, $\llcorner t_i\lrcorner_\circ=\mathsf{call}(\mathfrak{o}_i,D[\vec{\mathfrak{p}}_i])$ with $\psi(\mathfrak{o}_i)=\alpha^{j_i}$, for some $\mathfrak{o}_i,\vec{\mathfrak{p}}_i,j_i$, as required.
- If $x=\mathsf{ret}(D)$ then $\llcorner t_i\lrcorner_\circ=\llcorner t_i'\lrcorner_\circ\,\mathsf{ret}(D[\vec{\mathfrak{p}}_i])$ $(i=1,2)$, for some $\vec{\mathfrak{p}}_i$, and $t_i'\sqsubseteq t_i$ ending in last open call of t_i, and $\llcorner t_i\lrcorner_\psi=\llcorner t_i'\lrcorner_\psi\,\mathsf{ret}(D)$. The claim follows from the IH, and the fact that the $\vec{\mathfrak{p}}_i$'s are fresh (and Lemma 40).
- If $x=\underline{\mathsf{call}}(j,D[\vec{\alpha}])$ then $\llcorner t_i\lrcorner_\psi=\llcorner t_i'\lrcorner_\psi\,\underline{\mathsf{call}}(i,D[\vec{\alpha}])$, with $t_i=t_i'\,\underline{\mathsf{call}}(\mathfrak{p}_i,D[\vec{\mathfrak{o}}_i])$. Moreover, $\llcorner t_i\lrcorner_\circ=\llcorner t_i'\lrcorner_\circ\,\underline{\mathsf{call}}(\mathfrak{p}_i,D[\vec{\mathfrak{o}}_i])$. By definition $\mathfrak{p}_i$ is the j-th P-name in $\llcorner t_i'\lrcorner_\circ$. Then, the claim follows from the IH and the fact that the $\vec{\mathfrak{o}}_i$'s are fresh (and Lemma 40).
- If $x=\underline{\mathsf{ret}}(D[\vec{\alpha}])$ then work as the previous case above.

□

Lemma 44. *For each compatible* $\mathcal{A},\kappa,t$*:*

1) *the translation* $(\mathcal{A},\kappa,t)^\circ$ *is well defined;*
2) *if* $(\mathcal{A},\kappa,t)^\circ\ni(A,M,\hat{t},\hat{s},\psi)$ *then*

$$
\begin{aligned}
\hat{s} &= (t)_\psi^\top\\
M &= \{\llcorner t'\lrcorner_\psi \mid t'\sqsubseteq t\}\\
\hat{t} &= \begin{cases}\llcorner t\lrcorner_\psi & \text{if } |t| \text{ odd or } t=\cdot\\ \llcorner t'\lrcorner_\psi & \text{if } |t| \text{ even and } t'\sqsubseteq t \text{ ends in last open call of } t\end{cases}
\end{aligned}
$$

Proof. We use mutual induction on $|t|$. For 1, we need to check that the requirements imposed on M (when t ends in an O-move) are adhered to. Suppose WLOG that $t=t'+\underline{\mathsf{call}}(\mathfrak{p},D[\vec{\mathfrak{o}}])$ and let $(A',M',\hat{t}',\hat{s}',\psi')\in(\mathcal{A}',\kappa,t')^\circ$ (where $\mathcal{A}'$ the restriction of $\mathcal{A}$ to names in t'). If M is not defined on $\hat{t}'$ then (using also the IH) $(\mathcal{A},\kappa,t)^\circ$ is well defined. On the other hand, if $\mathsf{next}_M(\hat{t}')=x$ then we need to show that $x=\underline{\mathsf{call}}(j,D[\vec{\alpha}])$, where $\mathfrak{p}$ is the j-th P-name in $\llcorner t'\lrcorner_\circ$, for some $\vec{\alpha}$. Given $\mathsf{next}_M(\hat{t}')=x$, there is some $t''m\sqsubseteq t'$ such that $\llcorner t''m\lrcorner_\psi=\hat{t}'x$. We claim that $\llcorner t'\lrcorner\sim\llcorner t''\lrcorner$. By Lemma 43, $\llcorner t'\lrcorner_\circ\sim\llcorner t''\lrcorner_\circ$. Let $t_1=t',t_2=t''$ and suppose $\llcorner t_i\lrcorner_\circ$ starts with some $\mathsf{call}(\mathfrak{o}_i,D[\vec{\mathfrak{p}}_i])$ (for $i=1,2$). Since $\llcorner t_1\lrcorner_\psi=\llcorner t_2\lrcorner_\psi$, there are some α,j_1,j_2 such that $\psi(\mathfrak{o}_i)=\alpha^{j_i}$. Let $t_i'=t_i''o_i\sqsubseteq t_i$ each end in the O-move o_i that introduces $\mathfrak{o}_i$. If $t_1'=t_2'$ then we are done. Otherwise, the IH and the fact that ψ assigns the same name to $\mathfrak{o}_1,\mathfrak{o}_2$ imply that $\llcorner t_1''\lrcorner_\psi=\llcorner t_2''\lrcorner_\psi$ and, again by Lemma 43, we have that $\llcorner t_1''\lrcorner_\circ \sim \llcorner t_2''\lrcorner_\circ$. Moreover, using also the IH, we can obtain that $\llcorner t_1'\lrcorner_\circ \sim \llcorner t_2'\lrcorner_\circ$. Proceeding consecutively this way, we conclude that $\llcorner t'\lrcorner \sim \llcorner t''\lrcorner$. Hence, by O-innocence, $\llcorner t\lrcorner \sim \llcorner t''m\lrcorner$, which in turn implies that $\hat{s} = \underline{\mathsf{call}}(j, D[\vec{\alpha}])$.

We now look at 2. The base case is clear. Suppose that $t = t' + m$ and let $(A', M', \hat{t}', \hat{s}', \psi') \in (\mathcal{A}', \kappa', t')^\circ$. By IH, $M' = \{\llcorner t''\lrcorner_{\psi'} \mid t'' \sqsubseteq t'\}$ and therefore $\{\llcorner t''\lrcorner_\psi \mid t'' \sqsubseteq t\} = M' \cup \{\llcorner t\lrcorner_\psi\} = M'[\llcorner t\lrcorner_\psi]$. We do case analysis on m.

- Suppose $m = \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])$. By IH, $\hat{t}' = \llcorner t'\lrcorner_{\psi'}$. Moreover, $\llcorner t\lrcorner_\psi = \llcorner t'\lrcorner_\psi\, \underline{\mathsf{ret}}(D[\vec{\alpha}])$, with $\psi(\vec{\mathfrak{o}}) = \vec{\alpha}^{\cdots}$, and $M = M'[\hat{t}'\, \underline{\mathsf{ret}}(D[\vec{\alpha}])] = \{\llcorner t''\lrcorner_\psi \mid t'' \sqsubseteq t\}$. Also, $\hat{t}$ is the trace $\hat{t}''$ we obtain by translation by looking at the $t'' \sqsubseteq t$ ending in the last open call (which must be an O-move). By IH, $\hat{t}'' = \llcorner t''\lrcorner_\psi = \llcorner t\lrcorner_\psi$.
- Suppose $m = \underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$ By IH, $\hat{t}' = \llcorner t'\lrcorner_{\psi'}$. If t' is complete, then $\llcorner t\lrcorner_\psi = \cdot = \hat{t}$ and $M = M'$, so we are done. Otherwise, $\llcorner t\lrcorner_\psi = \llcorner t'\lrcorner_\psi\, \underline{\mathsf{call}}(i, D[\vec{\alpha}])$, with $\psi(\vec{\mathfrak{o}}) = \vec{\alpha}^{\cdots}$ and $\mathfrak{p}$ the i-th P-name in $\llcorner t'\lrcorner_\circ$, and $M = M'[\hat{t}'\, \underline{\mathsf{call}}(i, D[\vec{\alpha}])] = \{\llcorner t''\lrcorner_\psi \mid t'' \sqsubseteq t\}$. Also, $\hat{t} = \hat{t}'\, \underline{\mathsf{call}}(i, D[\vec{\alpha}]) = \llcorner t'\lrcorner_\psi\, \underline{\mathsf{call}}(i, D[\vec{\alpha}]) = \llcorner t\lrcorner_\psi$.
- Suppose $m = \mathsf{ret}(D[\vec{\mathfrak{p}}])$. If t is complete then the claim is clear. Otherwise, let $t'' \sqsubseteq t'$ end in the last open call in t' (which is an O-call), and $\hat{t}''$ the corresponding trace we obtain by translation. By IH, $\hat{t}'' = \llcorner t''\lrcorner_\psi$ and, hence, $\hat{t} = \hat{t}'\, \mathsf{ret}(D) \overset{\mathrm{IH}}{=} \llcorner t''\lrcorner_\psi\, \mathsf{ret}(D) = \hat{t}''\, \mathsf{ret}(D) = \llcorner t\lrcorner_\psi$. Moreover, $M = M'[\hat{t}'\, \mathsf{ret}(D)] = \llcorner t\lrcorner_\psi = \{\llcorner t''\lrcorner_\psi \mid t'' \sqsubseteq t\}$.
- Suppose $m = \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])$ and let $\psi'(\mathfrak{o}) = \alpha^{\cdots}$. Then, $\hat{t} = \mathsf{call}(\alpha, D) = \llcorner t\lrcorner_\psi$. Moreover, $M = M'[\mathsf{call}(\alpha, D)] = \{\llcorner t''\lrcorner_\psi \mid t'' \sqsubseteq t\}$.

The fact that $\hat{s} = (t)_\psi^\top$ follows from the IH using similar reasoning to the one above. $\square$

Given a play t and an O-name $\mathfrak{o}$ of t, let us write $t@\mathfrak{o}$ for the prefix $t' \sqsubseteq t$ such that the last move of t' is the one introducing $\mathfrak{o}$ in t.

Lemma 45. *Given a play t, even length $t_1, t_2 \sqsubseteq t$, names $\mathfrak{o}_1, \mathfrak{o}_2$ and some ψ from $(t)^\circ$ with $\psi(\mathfrak{o}_i) \doteq \alpha_i$ $(i = 1, 2)$:*

1) $\alpha_1 = \alpha_2$ iff $\llcorner t@\mathfrak{o}_1\lrcorner_\psi = \llcorner t@\mathfrak{o}_2\lrcorner_\psi \neq \varepsilon \vee t@\mathfrak{o}_1 = t@\mathfrak{o}_2$;
2) if $\llcorner t_1\lrcorner_\psi = \llcorner t_2\lrcorner_\psi \neq \varepsilon$ then $\llcorner t_1\lrcorner \sim \llcorner t_2\lrcorner$;
3) if $\alpha_1 = \alpha_2$ then $\llcorner t@\mathfrak{o}_1\lrcorner = \llcorner t@\mathfrak{o}_2\lrcorner$.

Proof. For 1, we use the definition of $(\_)^\circ$, and in particular the fact that ψ only assigns old names $\vec{\alpha}$ when the translated O-views are the same, and non-empty, in which case M is defined and forces O to play the same move. On the other hand, the translated O-views are empty only if the moves introducing $\mathfrak{o}_i$ are top-level, in which case the same α can be assigned two different moves, which means that the last moves in $t@\mathfrak{o}_i$ coincide, hence $t@\mathfrak{o}_1 = t@\mathfrak{o}_2$.
For 2, we do induction on $|t_1|+|t_2|$. If t_1 or t_2 is empty, the claim is clear. Otherwise, by Lemma 43 we have that $\llcorner t_1\lrcorner_\circ = \llcorner t_2\lrcorner_\circ$. Let p_i be the last open P-call in t_i and suppose it calls some $\mathfrak{o}_i$. By $\llcorner t_1\lrcorner_\psi = \llcorner t_2\lrcorner_\psi$ we have that ψ assigns the same name, say α, to $\mathfrak{o}_1, \mathfrak{o}_2$ and hence, by 1, we have that $\llcorner t@\mathfrak{o}_1\lrcorner_\psi = \llcorner t@\mathfrak{o}_2\lrcorner_\psi \neq \varepsilon$ or $t@\mathfrak{o}_1 = t@\mathfrak{o}_2$. If $\llcorner t@\mathfrak{o}_i\lrcorner_\psi \neq \varepsilon$ then, by IH, $\llcorner t@\mathfrak{o}_1\lrcorner = \llcorner t@\mathfrak{o}_2\lrcorner$ so, in either case, $\llcorner t@\mathfrak{o}_1\lrcorner = \llcorner t@\mathfrak{o}_2\lrcorner$. Observe that, for each i, $\llcorner t_i\lrcorner = \llcorner t@\mathfrak{o}_i\lrcorner \llcorner t_i\lrcorner_\circ$. The claim then follows by repeated application of Lemma 40.
Claim 3 follows from 1 and 2. $\square$

Suppose now we are given plays t_1, t_2, and consider $(M_i, \hat{t}_i, \hat{s}_i, \psi_i) \in (t_i)^\circ$ $(i = 1, 2)$. By a slight abuse of notation, we shall write $\psi_1 \triangleleft \psi_2$ if:

$$\forall \alpha, \mathfrak{o}_1.\ \psi_1(\mathfrak{o}_1) \doteq \alpha \implies \exists \mathfrak{o}_2.\ \psi_2(\mathfrak{o}_2) \doteq \alpha \wedge \llcorner t_1@\mathfrak{o}_1\lrcorner_{\psi_1} = \llcorner t_2@\mathfrak{o}_2\lrcorner_{\psi_2}.$$

We can show the following results.

Lemma 46. *Given $t_i' \sqsubseteq t_i$ $(i = 1, 2)$:*

1) if $M_1 \subseteq M_2$ then $\psi_1 \triangleleft \psi_2$;
2) if $\llcorner t_1'\lrcorner = \llcorner t_2'\lrcorner$ then $\llcorner t_1'\lrcorner_{\psi_1} \sim \llcorner t_2'\lrcorner_{\psi_2}$;
3) if $\llcorner t_1'\lrcorner = \llcorner t_2'\lrcorner$ and $\psi_1 \triangleleft \psi_2$ then $\llcorner t_1'\lrcorner_{\psi_1} = \llcorner t_2'\lrcorner_{\psi_2}$;
4) if $\llcorner t_1'\lrcorner = \llcorner t_2'\lrcorner$ and $M_1 \subseteq M_2$ then $\llcorner t_1'\lrcorner_{\psi_1} = \llcorner t_2'\lrcorner_{\psi_2}$.

Proof. For 1, suppose $M_1 \subseteq M_2$ and let $\psi_1(\mathfrak{o}_1) \doteq \alpha$ for some $\mathfrak{o}_1, \alpha$. Then, by definition, there is some $t_1'' \sqsubseteq t_1$ such that $\llcorner t_1''\lrcorner_{\psi_1} \in M_1$ and the last move in t_1'' is an O-move introducing $\mathfrak{o}_1$, and the corresponding name in $\llcorner t_1''\lrcorner_{\psi_1}$ is α. By $M_1 \subseteq M_2$ we have $\llcorner t_1''\lrcorner_{\psi_1} \in M_2$, so there is $t_2'' \sqsubseteq t_2$ such that $\llcorner t_1''\lrcorner_{\psi_1} = \llcorner t_2''\lrcorner_{\psi_2}$, so t_2'' ends in an O-move introducing some name $\mathfrak{o}_2$ and such that $\psi_2(\mathfrak{o}_2) \doteq \alpha$, as required.
For 2, we do induction on $|\llcorner t_1'\lrcorner| = |\llcorner t_2'\lrcorner|$. The base case is encompassed in that of t_1' being complete, in which case t_2' is also complete and the claim trivially holds. Similarly if $t_1' = t_1''m$ with t_1'' complete. Now let $t_i' = t_i''m$, with t_i', t_i'' not complete, and do case analysis on m. If m is a P-return then the claim follows from the IH; if m is a P-application, then it directly follows from the definition of $\llcorner t_i'\lrcorner_{\psi_i}$.
If m is an O-move then $\llcorner t_i'\lrcorner = \llcorner t_i''\lrcorner m$ and by IH $\llcorner t_1''\lrcorner_{\psi_1} \sim \llcorner t_2''\lrcorner_{\psi_2}$. If $m = \underline{\mathsf{call}}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$, so $\llcorner t_i'\lrcorner_{\psi_i}$ ends in some $\underline{\mathsf{call}}(j_i, D[\vec{\alpha}_i])$, by $\llcorner t_1'\lrcorner = \llcorner t_2'\lrcorner$ we obtain $j_1 = j_2$. As the $\vec{\alpha}_i$'s are fresh for $\llcorner t_i''\lrcorner_{\psi_i}$, from Lemma 40 we obtain $\llcorner t_1'\lrcorner_{\psi_1} \sim \llcorner t_2'\lrcorner_{\psi_2}$. Similarly if $m = \underline{\mathsf{ret}}(D[\vec{\mathfrak{o}}])$.

For 3, we again do induction on $|\llcorner t_1'\lrcorner| = |\llcorner t_2'\lrcorner|$. As above, let us assume that $t_i' = t_i''m$, with t_i', t_i'' not complete, and do a case analysis on m. If m is a P-return then we simply use the IH.
If $m = \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])$ then $\llcorner t_i'\lrcorner_{\psi_i} = \mathsf{call}(\alpha_i, D)$ with $\psi_i(\mathfrak{o}) \doteq \alpha_i$. Note that $\llcorner t_1'\lrcorner = \llcorner t_2'\lrcorner$ implies that $\llcorner t_1'@\mathfrak{o}\lrcorner = \llcorner t_2'@\mathfrak{o}\lrcorner$. Hence, by IH, $\llcorner t_1'@\mathfrak{o}\lrcorner_{\psi_1} = \llcorner t_2'@\mathfrak{o}\lrcorner_{\psi_2}$. As $\psi_1 \lhd \psi_2$, there is $\mathfrak{o}'$ such that $\psi_2(\mathfrak{o}') \doteq \alpha_1$ and $\llcorner t_1@\mathfrak{o}\lrcorner_{\psi_1} = \llcorner t_2@\mathfrak{o}'\lrcorner_{\psi_2}$, and therefore $\llcorner t_2@\mathfrak{o}\lrcorner_{\psi_2} = \llcorner t_2@\mathfrak{o}'\lrcorner_{\psi_2}$. By Lemma 45 we obtain that $\alpha_1 = \alpha_2$, as required.
If $m = \mathsf{ret}(D[\vec{\mathfrak{o}}])$ then $\llcorner t_i'\lrcorner_{\psi_i} = \llcorner t_i''\lrcorner_{\psi_i}\, \mathsf{ret}(D[\vec{\alpha}_i])$, with $\psi_i(\vec{\mathfrak{o}}) \doteq \vec{\alpha}_i$. By IH, $\llcorner t_1''\lrcorner_{\psi_1} = \llcorner t_2''\lrcorner_{\psi_2}$, so it suffices to show that $\vec{\alpha}_1 = \vec{\alpha}_2$. Let us pick some $\mathfrak{o}_j \in \vec{\mathfrak{o}}$, so $\psi_i(\mathfrak{o}_j) \doteq \alpha_{i,j}$. By hypothesis, there is $\mathfrak{o}_j'$ such that $\psi_2(\mathfrak{o}_j') \doteq \alpha_{1,j}$ and $\llcorner t_1''\lrcorner_{\psi_1} = \llcorner t_2@\mathfrak{o}_j'\lrcorner_{\psi_2}$. As $\llcorner t_1''\lrcorner_{\psi_1} = \llcorner t_2''\lrcorner_{\psi_2}$, we have $\llcorner t_2@\mathfrak{o}_j'\lrcorner_{\psi_2} = \llcorner t_2''\lrcorner_{\psi_2}$ so $\alpha_{1,j} = \alpha_{2,j}$, by Lemma 45. Similarly if $m = \mathsf{call}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$.
Claim 4 follows from 1 and 3. ☐

Lemma 47. *If $OV(t_1) \subseteq OV(t_2)$ then for each $(M_1, \hat{t}_1, \hat{s}_1, \psi_1) \in (t_1)^\circ$ there is $(M_2, \hat{t}_2, \hat{s}_2, \psi_2) \in (t_2)^\circ$ such that $\psi_1 \lhd \psi_2$ and $M_1 \subseteq M_2$.*

Proof. Note that $M_1 \subseteq M_2$ follows from $\psi_1 \lhd \psi_2$ and Lemmas 44 and 46, and the fact that $OV(t_1) \subseteq OV(t_2)$. We use induction on $|t_1|$ to show $\psi_1 \lhd \psi_2$. If t_1 is empty, then the claim is trivial. Otherwise, let $t_1 = t_1'm$ and let $(M_1', \hat{t}_1', \hat{s}_1', \psi_1') \in (t_1')^\circ$. Suppose $m = \mathsf{ret}(D[\vec{\mathfrak{o}}])$; the case where $m = \mathsf{call}(\mathfrak{p}, D[\vec{\mathfrak{o}}])$ is treated similarly. If M_1' is defined on $\hat{t}_1'$ then $\psi_1 = \psi_1'$ and the claim holds. Otherwise, $\psi_1 = \psi_1'[\vec{\mathfrak{o}} \mapsto \vec{\alpha}^i]$ for fresh $\vec{\alpha}$ and some i. By hypothesis, there is a permutation π and some $t_2'm \sqsubseteq \pi \cdot t_2$ such that $\llcorner t_1'\lrcorner = \llcorner t_2'\lrcorner$. By IH, there is $(M_2', \hat{t}_2', \hat{s}_2', \psi_2') \in (\pi \cdot t_2)^\circ$ with $\psi_1' \lhd \psi_2'$. Hence, by Lemma 46, $\llcorner t_1'\lrcorner_{\psi_1'} = \llcorner t_2'\lrcorner_{\psi_2'}$. Suppose $\psi_2'(\vec{\mathfrak{o}}) \doteq \vec{\alpha}'$, let $\pi' = (\vec{\alpha}\ \vec{\alpha}')$ and set $\psi_2 = \pi' \cdot \pi^{-1} \cdot \psi_2'$. Note that $\vec{\alpha}$ are fresh for ψ_1'. Suppose there is some $\alpha_j' \in \vec{\alpha}'$ that is not fresh for ψ_1', e.g. $\psi_1'(\mathfrak{o}_1') = \alpha_j'$ for some $\mathfrak{o}_1'$. We then must have some $\mathfrak{o}_2'$ such that $\psi_2'(\mathfrak{o}_2') \doteq \alpha_j'$ and $\llcorner t_1'@\mathfrak{o}_1'\lrcorner_{\psi_1'} = \llcorner(\pi \cdot t_2)@\mathfrak{o}_2'\lrcorner_{\psi_2'}$. By Lemma 45 we then obtain $\llcorner(\pi \cdot t_2)@\mathfrak{o}_2'\lrcorner_{\psi_2'} = \llcorner(\pi \cdot t_2)@\mathfrak{o}_j\lrcorner_{\psi_2'} = \llcorner t_2'\lrcorner_{\psi_2'}$. Recall that $\llcorner t_1'\lrcorner_{\psi_1'} = \llcorner t_2'\lrcorner_{\psi_2'}$, hence $\llcorner t_1'\lrcorner_{\psi_1'} = \llcorner t_1'@\mathfrak{o}_1'\lrcorner_{\psi_1'}$. But then, by Lemma 45 again, we have that $\alpha_j' = \alpha_j$, a contradiction to α_j being fresh. Hence, $\vec{\alpha}'$ are fresh for ψ_1', and thus $\psi_1' = \pi' \cdot \psi_1'$, which implies $\psi_1' \lhd \pi' \cdot \psi_2$ and therefore $\psi_1 \lhd \pi' \cdot \psi_2$. Since π only permutes names from the extended LTS, we can also deduce that $\psi_1 \lhd \pi^{-1} \cdot \pi' \cdot \psi_2 = \psi_2$. Finally, from $(M_2', \hat{t}_2', \hat{s}_2', \psi_2') \in (\pi \cdot t_2)^\circ$ we obtain that $(M_2', \hat{t}_2', \hat{s}_2', \pi^{-1} \cdot \psi_2') \in (t_2)^\circ$ and thus $(\pi' \cdot M_2', \pi' \cdot \hat{t}_2', \pi' \cdot \hat{s}_2', \psi_2) \in (t_2)^\circ$, as required.
If m is a P-move, then the claim follows directly from the IH (on t_1'). ☐

Lemma 48. *Given $(M_i, \hat{t}_i, \hat{s}_i, \psi_i) \in (t_i)^\circ$ (for $i = 1, 2$), if $M_1 \subseteq M_2$ and $\hat{s}_1 = \hat{s}_2$ then $OV(t_1) \subseteq OV(t_2)$.*

Proof. We show that, for any $t_1' \sqsubseteq t_1$, we have $OV(t_1') \subseteq OV(t_2)$, by induction on $|t_1'|$. Base case is clear.
Now let $t_1' = t_1''m_1$ with m_1 an O-move so, by IH, we have $OV(t_1'') \subseteq OV(t_2)$. We do case analysis on m_1. Suppose $m_1 = \mathsf{call}(\mathfrak{p}_1, D[\vec{\mathfrak{o}}_1])$ and $\psi_1(\vec{\mathfrak{o}}_1) \doteq \vec{\alpha}$, and assume t_1'' is not complete; the case of $m_1 = \mathsf{ret}(D[\vec{\mathfrak{o}}_1])$ is dealt with similarly. Then, by definition, $\mathsf{next}_{M_1}(\llcorner t_1''\lrcorner_{\psi_1}) = \mathsf{call}(j, D[\vec{\alpha}])$ and, since $M_1 \subseteq M_2$, $\mathsf{next}_{M_2}(\llcorner t_1''\lrcorner_{\psi_1}) = \mathsf{call}(j, D[\vec{\alpha}])$. By IH we have $\llcorner t_1''\lrcorner \in OV(t_2)$, so let $t_2'' \sqsubseteq \pi \cdot t_2$ (for some π) such that $\llcorner t_1''\lrcorner = \llcorner t_2''\lrcorner$. By Lemma 46, $\llcorner t_1''\lrcorner_{\psi_1} = \llcorner t_2''\lrcorner_{\pi\cdot\psi_2}$. By hypothesis, $\mathsf{next}_{M_2}(\llcorner t_2''\lrcorner_{\pi\cdot\psi_2}) = \mathsf{call}(j, D[\vec{\alpha}])$, so there is m_2 such that $t_2''m_2 \sqsubseteq \pi \cdot t_2$ and $\llcorner t_2''m_2\lrcorner_{\pi\cdot\psi_2} = \llcorner t_1''\lrcorner_{\psi_1}\, \mathsf{call}(j, D[\vec{\alpha}])$, hence $m_2 = \mathsf{call}(\mathfrak{p}_2, D[\vec{\mathfrak{o}}_2])$. By definition, $\mathfrak{p}_i$ is the j-th P-name in $\llcorner t_i''\lrcorner$, thus $\mathfrak{p}_1 = \mathfrak{p}_2$. Using also Lemma 40, we have $\llcorner t_1''\lrcorner m_1 \sim \llcorner t_2''\lrcorner m_2$, as required. Finally, suppose t_1'' is complete and pick $t_2'' \sqsubseteq \pi \cdot t_2$ (for some π) such that $\llcorner t_1''\lrcorner = \llcorner t_2''\lrcorner$. By $\hat{s}_1 = \hat{s}_2$ and Lemma 44, and since $(M_2, \hat{t}_2, \hat{s}_2, \pi \cdot \psi_2) \in (\pi \cdot t_2)^\circ$, there is m_2 such that $t_2''m_2 \sqsubseteq \pi \cdot t_2$ and $m_2 = \mathsf{call}(\mathfrak{p}_2, D[\vec{\mathfrak{o}}_2])$, and we can conclude as above.
Suppose t_1 ends in $m_1 = \mathsf{call}(\mathfrak{o}, D[\vec{\mathfrak{p}}])$ and let $t_1'' = t_1'@\mathfrak{o} \sqsubseteq t_1'$. By IH, there is $t_2'' \sqsubseteq \pi \cdot t_2$ (for some π) such that $\llcorner t_1''\lrcorner = \llcorner t_2''\lrcorner$, so in particular t_2'' ends in o. By Lemma 46, $\llcorner t_1''\lrcorner_{\psi_1} = \llcorner t_2''\lrcorner_{\pi\cdot\psi_2}$, hence there is some α such that $\psi_1(\mathfrak{o}_1) \doteq \alpha$ and $(\pi \cdot \psi_2)(\mathfrak{o}_2) \doteq \alpha$. Since $M_1 \subseteq M_2$, we have $\mathsf{call}(\alpha, D) \in M_2$ and hence there must be $s_2'm_2 \sqsubseteq \pi \cdot t_2$ with $m_2 = \mathsf{call}(\mathfrak{o}', D[\vec{\mathfrak{p}}'])$ such that $(\pi \cdot \psi_2)(\mathfrak{o}') \doteq \alpha$. Hence, by Lemma 45, $\llcorner t_1''\lrcorner = \llcorner t_2''\lrcorner = \llcorner s_2'@\mathfrak{o}'\lrcorner$. Using Lemma 40, we obtain $\llcorner t_1''\lrcorner m_1 \sim \llcorner s_2'@\mathfrak{o}'\lrcorner m_2$, as required. If t_1 ends in some $\mathsf{ret}(D)\vec{\mathfrak{p}}$ then we work similarly, using also the fact that $\hat{s}_1 = \hat{s}_2$ in case m_1 is top-level. ☐

Definition 49. We define a translation of configurations from the game-LTS to the plain LTS as follows:

$$(\langle \mathcal{A} ; \kappa ; K ; t ; V ; \vec{\mathfrak{p}} \rangle)^\circ = \{\langle A ; M ; \hat{K} ; \hat{t} ; \hat{V} ; \vec{v} \rangle \mid \exists\psi.\ (A, \hat{M}, \hat{t}, \hat{s}, \psi) \in (\mathcal{A}, \kappa, t)^\circ\}$$
$$(\langle \mathcal{A} ; \kappa ; K ; t ; e ; V ; \vec{\mathfrak{o}} \rangle)^\circ = \{\langle A ; M ; \hat{K} ; \hat{t} ; \hat{e} ; \hat{V} \rangle \mid \exists\psi.\ (A, \hat{M}, \hat{t}, \hat{s}, \psi) \in (\mathcal{A}, \kappa, t)^\circ\}$$

where $(\hat{e}, \hat{K}, \hat{V}, \vec{v}) = \psi(e, K, \kappa(V), \kappa(\vec{\mathfrak{p}}))$. ☐

We can now show the following result. Below we write x for τ or a move m, and χ for τ or a move η.

Lemma 50. *Let $\mathcal{C}$ be a reachable configuration of the game-LTS and let $C \in (\mathcal{C})^\circ$. Then:*

- *if $\mathcal{C} \xrightarrow{x} \mathcal{C}'$ then $\exists C' \in (\mathcal{C}')^\circ, \chi.\ C \xrightarrow{\chi} C'$;*
- *if $C \xrightarrow{\chi} C'$ then $\exists \mathcal{C}', x.\ \mathcal{C} \xrightarrow{x} \mathcal{C}' \wedge C' \in (\mathcal{C}')^\circ$;*

and either $\chi = x = \tau$, or χ, x are of the same O/P and call/return shape, with their components related as in $C, \mathcal{C}$ and $C', \mathcal{C}'$.

Recall that we write $\mathcal{C}_e$ for the initial configuration for term $\vdash e : T$ in the game-LTS, and C_e for the one in the plain LTS.

Proposition 51. *Given* $\vdash e_1, e_2 : T$*,* $OV_{tl}(\mathcal{C}_{e_1}) = OV_{tl}(\mathcal{C}_{e_2})$ *iff* $[\![e_1]\!] = [\![e_2]\!]$*.*

Proof. Suppose that $OV_{tl}(\mathcal{C}_{e_1}) = OV_{tl}(\mathcal{C}_{e_2})$, and let $(t_1, M_1) \in [\![e_1]\!]$. We show that $(t_1, M_1) \in [\![e_2]\!]$. Let $C_{e_1} \xrightarrow{t_1}\!\!\!\!\twoheadrightarrow C'_{e_1}$, with $C'_{e_1} = \langle A\,;M_1\,;\varepsilon\,;\hat{t}_1\,;V\,;\vec{v}\rangle$. By Lemma 50, $\mathcal{C}_{e_1} \xrightarrow{s_1}\!\!\!\!\twoheadrightarrow \mathcal{C}'_{e_1}$ with $(\mathcal{C}'_{e_1})^\circ \ni C'_{e_1}$. By hypothesis, $\mathcal{C}_{e_2} \xrightarrow{s_2}\!\!\!\!\twoheadrightarrow \mathcal{C}'_{e_2}$ with $OV(s_1) = OV(s_2)$, and s_1, s_2 both top-linear. WLOG, assume that $\llcorner s_1 \lrcorner = \llcorner s_2 \lrcorner$. By Lemma 50 again, $C_{e_2} \xrightarrow{t_2}\!\!\!\!\twoheadrightarrow C'_{e_2}$ with $C'_{e_2} = (\mathcal{C}'_{e_2})^\circ$. By Lemma 47, the last claim of Lemma 50 and the fact that $\llcorner s_1 \lrcorner = \llcorner s_2 \lrcorner$, we can pick C'_{e_2} in such a way that its M-component is M_1 and $t_1 = t_2$.

Conversely, suppose $[\![e_1]\!] = [\![e_2]\!]$ and let t_1 be a complete (top-linear) play of $\mathcal{C}_{e_1}$. We need to show $OV(t_1) \in OV_{tl}(\mathcal{C}_{e_2})$. Considering the transition sequence $\mathcal{C}_{e_1} \xrightarrow{t_1}\!\!\!\!\twoheadrightarrow \mathcal{C}_1$, by Lemma 50 there is $C_{e_1} \xrightarrow{\hat{t}}\!\!\!\!\twoheadrightarrow C_1$ with $C_1 \in (\mathcal{C}_1)^\circ$ and final M-component M. By hypothesis then, $C_{e_2} \xrightarrow{\hat{t}}\!\!\!\!\twoheadrightarrow C_2$ with final M-component M. Using Lemma 50 again we obtain $\mathcal{C}_{e_2} \xrightarrow{t_2}\!\!\!\!\twoheadrightarrow \mathcal{C}_2$ with $C_2 \in (\mathcal{C}_2)^\circ$ and such that $\hat{t}$ is the corresponding translation of both t_1 and t_2 (so t_2 also top-linear). Then, by Lemma 48, we have $OV(t_1) = OV(t_2)$. $\square$