# Infinite Trace Equivalence

Paul Blain Levy

No Institute Given

**Abstract.** We solve a longstanding problem by providing a denotational model for nondeterministic programs that identifies two programs iff they have the same range of possible behaviours. We discuss the difficulties with traditional approaches, where divergence is bottom or where a term denotes a function from a set of environments. We see that making forcing explicit, in the manner of game semantics, allows us to avoid these problems.
We begin by modelling a first-order language with sequential I/O and unbounded nondeterminism (no harder to model, using this method, than finite nondeterminism). Then we extend the semantics to higher-order and recursive types by adapting earlier game models. Traditional adequacy proofs using logical relations are not applicable, so we use instead a novel hiding argument.

## 1 Introduction

### 1.1 The Problem

Consider the following (call-by-name) language of nondeterministic commands:

$$M ::= \quad \mathtt{x} \mid \mathtt{print}\, c.\, M \mid \mu \mathtt{x}.M \mid M \,\mathtt{or}\, M$$

where $\mu$ is recursion, and $c$ ranges over some alphabet $\mathcal{A}$. A closed term can behave in two ways: to print finitely many characters and then diverge, or to print infinitely many characters. Two closed terms are said to be *infinite trace equivalent* when they have the same range of possible behaviours.

As stated in [Plo83], "we [...] desire a semantics such that $\mathcal{C}[\![c]\!](\sigma)$ is the set of tapes that might be output", i.e. a model whose kernel on closed terms is infinite trace equivalence. Some models of nondeterminism, such as the various powerdomains [Plo83] and divergence semantics [Ros04], identify programs that are not infinite trace equivalent, so they are too coarse. Others count the internal manipulations [Bro02, Esc98] or include branching-time information, so they are too fine (at best) for this problem.

In this paper, we provide a solution, and see that it can be used to model not only the above language, but also unbounded nondeterminism, input (following a request), and higher-order, sum and recursive types. Our model is a form of pointer game semantics [HO00], although the technology of pointer games is only needed for the higher-order types. This gives a good illustration of the power and flexibility of game semantics.

Proving the computational adequacy of the model incorporating higher-order, sum and recursive types presents a difficulty, because the traditional method, using a logical relation, is not applicable to it. So we give, instead, a proof that uses the method of *hiding*. As a byproduct, we obtain a very simple proof of the adequacy of the game model of FPC [McC96].

### 1.2 Why Explicit Forcing?

Before turning to our solution, we consider two kinds of semantics that have been studied. In both cases, suppose the alphabet is singleton $\{\checkmark\}$.

1. A *divergence-least* semantics is one where a term denotes an element of a poset, every construct is monotone, and $[\![\mu \mathtt{x}.\mathtt{x}]\!]$ denotes a least element $\bot$. Examples are the Hoare, Smyth and Plotkin powerdomain semantics [Plo83], all the CSP semantics in [Ros98], and the game semantics of [HM99]. Divergence-least semantics cannot model infinite trace equivalence, by the following argument taken from [Plo83].

   | | | |
   |---|---|---|
   | Put | $M = \bot \mathtt{\ or\ } \checkmark.\checkmark.\bot$ | $M' = \bot \mathtt{\ or\ } \checkmark.\bot \mathtt{\ or\ } \checkmark.\checkmark.\bot$ |
   | Then | $M = \bot \mathtt{\ or\ } \bot \mathtt{\ or\ } \checkmark.\checkmark.\bot \leqslant M' \leqslant \bot \mathtt{\ or\ } \checkmark.\checkmark.\bot \mathtt{\ or\ } \checkmark.\checkmark.\bot \leqslant M$ | |

   Hence $M = M'$, contradicting infinite trace equivalence.
2. A *well-pointed* semantics is one where (roughly speaking) a term (or at least a closed term) denotes a function from the set of *environments*. Examples are the 3 powerdomain semantics [Plo83], all the CSP semantics in [Ros98], the semantics using infinite traces in [Bro02], and divergence semantics [Ros04]. In general, well-pointed semantics ars appropriate for equivalences satisfying the *context lemma* property: terms equivalent in every environment are equivalent in every context. However, infinite trace equivalence does not satisfy this property, as the following two terms[1] involving $\mathtt{x}$ demonstrate:

   $$N = \mathtt{x} \mathtt{\ or\ } \mu\mathtt{y}.(\checkmark.\mathtt{y} \mathtt{\ or\ } \mu\mathtt{z}.\mathtt{z}) \qquad\qquad N' = \checkmark.\mathtt{x} \mathtt{\ or\ } \mu\mathtt{y}.(\checkmark.\mathtt{y} \mathtt{\ or\ } \mu\mathtt{z}.\mathtt{z})$$

   Clearly $\mu\mathtt{x}.N'$ can print $\checkmark^\omega$ whereas $\mu\mathtt{x}.N$ cannot, so any model of infinite trace equivalence must distinguish $N$ from $N'$. But $N[M/\mathtt{x}]$ and $N'[M/\mathtt{x}]$ are infinite trace equivalent for every closed term $M$.

Naively, $N$ and $N'$ can be distinguished by saying that $N'$ is able to print a tick and then force (i.e. execute) $\mathtt{x}$, whereas $N$ is not. And that gives our solution.

This idea, that a model should make explicit when a call-by-name program forces its (thunked) argument, is present (often implicitly) in game semantics, where (as argued in [Lev04]) "asking a question" indicates forcing a thunk. That is why our solution fits into the game framework. However, the game models in the literature are divergence-least, and this property is exploited by adequacy proofs using logical relations. This is even true of the nondeterministic model of [HM99], where strategy sets are quotiented by the Egli-Milner preorder[2] and so they become cpos. The novelty of this paper is that it avoids such quotienting.

### 1.3 Structure Of Paper

We extend the language of Sect. 1.1 in three stages.

[1] discovered by A. W. Roscoe in 1989 (personal communication), and independently in [Levb].

[2] The quotienting is needed there to make composition associative, because strategies are not defined to include infinite traces.

Firstly, in Sect. 2.1, we bring in erratic (aka internal) choice operators of arbitrary arity, which compels us to consider finite traces as well as infinite traces.

Secondly, in Sect. 2.2, we add *requested input*, for example printing a request such as `Please enter your name`, then waiting for the user to enter a string. This kind of I/O is familiar to beginning programmers. At this stage we can still give a non-technical denotational semantics—we do that in Sect. 2.4.

The third extension, in Sect. 4, is higher-order and recursive types. Before doing this, we introduce the basic structures of pointer games in Sect. 3, which we use in the model.

**Acknowledgements**

I thank Martín Escardó and Guy McCusker—both of whom showed me adequacy proofs that count execution steps—and Russ Harmer and Bill Roscoe, for helpful information.

## 2 First-Order Language

### 2.1 Erratic Choice and Omni-Errors

To allow choice operators of various arities, we define:

**Definition 1**
An *erratic signature* $Y$ is a family of sets $\{P_h\}_{h\in H}$, together with a set $U$, such that either all $P_h$ are nonempty or $U$ is nonempty. □

Any such $Y$—together with an alphabet $\mathcal{A}$—determines a language $\mathcal{L}(Y,\mathcal{A})$ in which

- each $h \in H$ provides an erratic choice operator $\mathtt{choose}^h$ of arity $P_h$
- each $u \in U$ is an *omni-error*, meaning that *any* program, at any time, is allowed to abort, printing `Omni-error message` $u$.

The syntax is

$$M ::= \quad \mathtt{x} \mid \mathtt{print}\ c.\ M \mid \mu\mathtt{x}.M \mid \mathtt{choose}^h_{p\in P_h}\ M_p$$

For each context $\Gamma = \mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$, we define a terminable LTS $\mathcal{L}(Y,\mathcal{A},\Gamma)$ with labels $\mathcal{A}\cup\{\tau\}$. Its states are the terms $\Gamma \vdash M$, and its terminal states are the free identifiers. The transitions are

$$\mu\mathtt{x}.M \overset{\tau}{\rightsquigarrow} M[\mu\mathtt{x}.M/\mathtt{x}] \qquad \mathtt{choose}^h_{p\in P_h}\ M_p \overset{\tau}{\rightsquigarrow} M_{\hat{p}} \quad (\hat{p}\in P_h)$$
$$\mathtt{print}\ c.\ M \overset{c}{\rightsquigarrow} M$$

and we also write $M \overset{u}{\rightsquigarrow}$ for every closed term $M$ and $u \in U$.

Usually, $U$ would be empty, so omni-errors cannot happen. But if $P_h$ is empty, for some $h \in H$, then the program $\mathtt{choose}^h$ has no way of behaving other than to raise an omni-error. And if $U$ is empty too, then there is no way at all for the program to behave (deadlock); so the language cannot be implemented. Rather than bring up the issue of failures, Def. 1 simply excludes this situation.

A program in this language can behave in 3 ways:

1. print finitely many characters then diverge
2. print infinitely many characters
3. print finitely many characters then raise an omni-error.

For a closed term $M$, let us write $[M]_{\mathsf{inf}} \subset \mathcal{A}^* \times \mathcal{A}^\omega$ for the range of behaviours of type (1)–(2) that $M$ can exhibit. Let us write $[M]_{\mathsf{fin}} \subset \mathcal{A}^*$ for the set of finite traces (without omni-error) of $M$, and write $[M]$ for the pair $([M]_{\mathsf{fin}}, [M]_{\mathsf{inf}})$. The range of behaviours of type (3) that $M$ can exhibit is precisely $[M]_{\mathsf{fin}} \times U$. So the complete range of behaviours of $M$ is $[M]_{\mathsf{inf}} + [M]_{\mathsf{fin}} \times U$, which we call $[M]_U$. We define *infinite trace equivalence* to be the kernel of $[-]_U$.

**Proposition 1** 1. For some signatures and alphabets, infinite trace equivalence is strictly finer than the kernel of $[-]_{\mathsf{inf}}$.
2. For all signatures and alphabets, infinite trace equivalence is the kernel of $[-]$.

□

*Proof.* (1) Consider $\checkmark.\mathtt{choose}^h$ and $\mathtt{choose}^h$, where $P_h$ is empty. (2) By the nonemptiness assumption, every finite path extends to a path that is either infinite or ends in an omni-error. □

Henceforth we regard Prop. 1(2) as *defining* infinite trace equivalence, and ignore omni-errors.

### 2.2 Requested Input

For the second extension (see Sect. 1.3), we define an *I/O signature* to be a family of sets $Z = \{I_o\}_{o \in O}$. Each $o \in O$ provides a requested input operator $\mathtt{input}^o$ of arity $I_o$, that prints $o$ and then waits for the user to supply some $i \in I_o$. We say $Z$ is *countable* when $O$ and each $I_o$ is countable.

Given a signature $Z$, we write $R_Z$ for the endofunctor on $\mathbf{Set}$ mapping $X$ to $\sum_{o \in O} X^{I_o}$. We then obtain a strong monad $T_Z$ on $\mathbf{Set}$ (the free monad on $R_Z$) mapping $A$ to $\mu Y.(A + R_Z Y)$. This monad can be used, in the manner of [Mog91, PP02], to model requested input. Note that this includes as special cases the monads designated in [Mog91] as "interactive input", "interactive output", and "exceptions". We accordingly regard each output $c \in \mathcal{A}$ as an element of $O$ such that $I_o = 1$, and we regard $\mathtt{print}\ c.\ M$ as syntactic sugar[3] for $\mathtt{input}^c_{i \in 1}\ M$.

### 2.3 Bi-Labelled Transition Systems

To describe the behaviour of a system using requested input, the following variation on an LTS is most natural.

**Definition 2** (BLTS)

1. A *bi-labelled transition system* (BLTS) $\mathcal{L}$, wrt an I/O signature $Z = \{I_o\}_{o \in O}$, consists of

[3] The operational difference between these constructs appears to be denotationally immaterial, at least in the sequential setting.

- a set $S$ of *states*, each of which is classified as either a *silent state* or an *o-state* for some request $o \in O$—we write $S_{\mathsf{sil}}$ and $S_o$ for the set of silent states and of $o$-states, respectively
- a relation $\rightsquigarrow$ from $S_{\mathsf{sil}}$ to $S$, and, for each $o \in O$, a function $S_z \times I_o \xrightarrow{\cdot\,:\,} S$

More abstractly, it is a coalgebra for the endofunctor $\mathcal{P} + R_Z$ on $\mathbf{Set}$.

2. A *terminable BLTS* is the same, except that there can be *terminal* states.
3. A terminable BLTS, is *deterministic* when each silent state has precisely one successor.
4. Given a terminable BLTS $\mathcal{L}$ wrt $Z + Z'$, the *hiding* of $\mathcal{L}$, written $\mathcal{L} \upharpoonright Z$ is the BLTS wrt $Z$ obtained as follows: each $o$-state $d$ (where $o \in Z'$) becomes silent, and $d \rightsquigarrow d'$ if $d : i = d'$ for some $i \in I_o$.

□

As with LTS's, we can obtain trace sets of states, in the following way.

**Definition 3** (strategies) Let $Z$ be an I/O signature, and let $V$ be a set.

1. An *play* wrt $Z$ is a finite or infinite sequence $o_0 i_0 o_1 i_1 \ldots$ where $o_r \in O$ and $i_r \in I_{o_r}$. It *awaits Player* if of even length, and *awaits o-input* if of odd length ending in $o$. A *play terminating in* $V$ wrt $Z$ is a Player-awaiting play extended with an element of $V$.
2. A *nondeterministic infinite trace* (NIT) *strategy into* $V$ consists of
   **finite traces** a set $A$ of input-awaiting plays
   **divergences** a set $B$ of Player-awaiting plays
   **infinite traces** a set $C$ of infinite plays
   **terminating traces** a set $D$ of plays terminating in $V$
   such that if $s$ is in $A$, $B$, $C$ or $D$, then every input-awaiting prefix of $s$ is in $A$. We write $T_Z^{\mathsf{NIT}}\ V$ for the set of all NIT strategies into $V$. Clearly $T_Z^{\mathsf{NIT}}$ is a strong monad on $\mathbf{Set}$.
3. A Player-awaiting or infinite play is *consistent* with a strategy $\sigma$ when every input-awaiting prefix is a finite trace of $\sigma$.
4. For $d \in V$, we define $\eta d$ (the monad's unit at $d$) to be the strategy

$$(\{\}, \{\}, \{\}, \{d\})$$

5. Given a family of strategies $\{\sigma_i\}_{i \in I}$, where $\sigma_i = (A_i, B_i, C_i, D_i)$, we write $\bigcup_{i \in I} \sigma_i$ for the strategy

$$(\bigcup_{i \in I} A_i, \bigcup_{i \in I} B_i, \bigcup_{i \in I} C_i, \bigcup_{i \in I} D_i)$$

6. Given $o \in O$, and for each $i \in I_o$ a strategy $\sigma_i = (A_i, B_i, C_i, D_i)$, we write $\mathsf{input}^o_{i \in I_o}\ \sigma_i$ for the strategy

$$(\{o\} \cup \{ois | i \in I_o, s \in A_i\}, \{ois | i \in I_o, s \in B_i\}, \{ois | i \in I_o, s \in C_i\}, \{ois | i \in I_o, s \in D_i\})$$

7. A strategy $(A, B, C, D)$ is *deterministic* when
   - any Player-awaiting play consistent with it has at most one immediate extension to a play in $A$ or $D$, and is in $B$ iff it has no such extension

– any infinite play consistent with it is in $C$.

8. Given a Player-awaiting (resp. infinite) play $s$ wrt $Z + Z'$, we write $s \upharpoonright Z$ for the Player-awaiting (resp. Player-awaiting or infinite) play obtained by removing moves in $Z'$.
9. Given a strategy $\sigma = (A, B, C, D)$ into $V$ wrt $Z + Z'$, the *hiding* of $\sigma$, written $\sigma \upharpoonright Z$, is the strategy wrt $Z$ given by

$$\begin{gathered}(\{(s \upharpoonright Z)o | so \in A, o \in Z\},\\ \{s \upharpoonright Z | s \in B\} \cup \{s \upharpoonright Z | s \in C, s \upharpoonright Z \text{ awaiting Player}\},\\ \{s \upharpoonright Z | s \in C, s \upharpoonright Z \text{ infinite}\},\\ \{(s \upharpoonright Z)v | sv \in D\})\end{gathered}$$

□

**Proposition 2** 1. The strategy $\eta z$ is deterministic, and $\mathsf{input}^o$ preserves determinism.

2. Given signatures $Z$ and $Z'$, the hiding of

$$\begin{aligned}\eta v \text{ is } & \eta v\\ \bigcup_{i \in I} \sigma_i \text{ is } & \bigcup_{i \in I} (\sigma_i \upharpoonright Z)\\ \mathsf{input}^o_{i \in I_o}\ \sigma_i \text{ is } & \begin{cases} \mathsf{input}^o_{i \in I_o}\ (\sigma_i \upharpoonright Z) & \text{if } o \in Z \\ \bigcup_{i \in I_o} (\sigma_i \upharpoonright Z) & \text{if } o \in Z' \end{cases}\end{aligned}$$

where each $\sigma_i$ is a strategy wrt $Z + Z'$.

□

**Definition 4** (BLTS to strategies) Let $Z$ be an I/O signature, and let $\mathcal{L}$ be a terminable BLTS wrt $Z$. Write $V$ for its set of terminal states.

1. For each state $d \in S$, we write $[d]_{\mathcal{L}}$, or just $[d]$, for the NIT strategy $(A, B, C, D)$ into $V$ where an input awaiting play $so$ (respectively divergence $s$, infinite play $s$, terminating trace $sv$) is in $A$ (resp. $B, C, D$) iff there is a sequence of transitions from $d$ to some $o$-state (resp. infinite sequence from $d$, infinite sequence from $d$, sequence of transitions from $d$ to $v$) whose sequence of non-silent actions is $s$.
2. Two states $d$ and $d'$ are *infinite trace equivalent* when $[d] = [d']$.

□

**Proposition 3** For any state $d$ in a terminable BLTS, $[d]$ is $\mathsf{input}^o_{i \in I_o}\ [d : i]$ or $\bigcup_{d \rightsquigarrow d'} [d']$ or $\eta d$ according as $d$ is an $o$-state or silent or terminal. □

### 2.4 Operational and Denotational Semantics

Now we are in a position to treat the second extension (see Sect. 1.3). An erratic signature $Y = \{P_h\}_{h \in H}$ and I/O signature $Z = \{I_o\}_{o \in I}$ define a language $\mathcal{L}(Y, Z)$ whose syntax is given by

$$M ::= \quad \mathtt{x} \mid \mu \mathtt{x}.M \mid \mathtt{choose}^h_{p \in P_h}\ M_p \mid \mathtt{input}^o_{i \in I_o}\ M_i$$

Each context gives rise to a terminable BLTS wrt $Z$, called $\mathcal{L}(Y, Z, \Gamma)$. Its states are the terms $\Gamma \vdash M$, and its terminal states are the free identifiers. $\mu\mathtt{x}.M$ and $\mathtt{choose}^h_{p\in P_h}\ M_p$ are silent, and $\mathtt{input}^o_{i\in I_o}\ M_i$ is an $o$-state. The transitions are

$$\mu\mathtt{x}.M \rightsquigarrow M[\mu\mathtt{x}.M/\mathtt{x}] \quad \mathtt{choose}^h_{p\in P_h}\ M_p \rightsquigarrow M_{\hat{p}} \text{ for each } \hat{p} \in P_h$$
$$(\mathtt{input}^o_{i\in I_o}\ M_i) : \hat{\imath} = M_{\hat{\imath}} \text{ for each } \hat{\imath} \in I_o$$

These transition systems have the following properties.

**Lemma 1** Suppose $\Gamma, \mathtt{x} \vdash M$ and $\Gamma \vdash N$. Suppose that $M$ is not $\mathtt{x}$.

1. $M$ is silent iff $M[N/\mathtt{x}]$ is. If, moreover, $M \rightsquigarrow M'$ then $M[N/\mathtt{x}] \rightsquigarrow M'[N/\mathtt{x}]$. Conversely, if $M[N/\mathtt{x}] \rightsquigarrow Q$ then $M \rightsquigarrow M'$ for some $M'$ such that $Q = M'[N/\mathtt{x}]$.
2. $M$ is an $o$-state iff $M[N/\mathtt{x}]$ is, and then $M[N/\mathtt{x}] : i = (M : i)[N/\mathtt{x}]$ for each $i \in I_o$.
3. For each $\mathtt{y} \in \Gamma$, we have $M = \mathtt{y}$ iff $M[N/\mathtt{x}] = \mathtt{y}$.

□

**Proposition 4** 1. $[\mathtt{x}] = \eta\mathtt{x}$
2. $[\mathtt{choose}^h_{p\in P_h}\ M_p] = \bigcup_{p\in P_h}[M_p]$
3. $[\mathtt{input}^o_{i\in I_o}\ M_i] = \mathsf{input}^o_{i\in I_o}\ [M_o]$
4. If $\Gamma, \mathtt{x} \vdash M$ then $[\mu\mathtt{x}.M] = \mu[M]$, where we define $\mu(A, B, C, D)$ to be

$$\begin{aligned}(\{l_0 \cdots l_{n-1} lo | l_0\mathtt{x}, \ldots, l_{n-1}\mathtt{x} \in D, lo \in A\},\\ \{l_0 \cdots l_{n-1} l | l_0\mathtt{x}, \ldots, l_{n-1}\mathtt{x} \in D, l \in B\} \cup \{l_0 \cdots l_{n-1} | l_0\mathtt{x}, \ldots, l_{n-1}\mathtt{x} \in D, \mathtt{x} \in D\},\\ \{l_0 \cdots l_{n-1} l | l_0\mathtt{x}, \ldots, l_{n-1}\mathtt{x} \in D, l \in C\} \cup \{l_0 l_1 \cdots | l_0\mathtt{x}, l_1\mathtt{x}, \ldots \in D\},\\ \{l_0 \cdots l_{n-1} l\mathtt{y} | l_0\mathtt{x}, \ldots, l_{n-1}\mathtt{x} \in D, l\mathtt{y} \in D\})\end{aligned}$$

5. If $\Gamma, \mathtt{x} \vdash M$ and $\Gamma \vdash N$, then $[M[N/\mathtt{x}]]$ is $[M]*[N]$, where we define $(A, B, C, D)*(A', B', C', D')$ to be

$$\begin{aligned}(A \cup \{ll'o | l\mathtt{x} \in D, l'o \in A'\},\\ B \cup \{ll' | l\mathtt{x} \in D, l' \in B'\},\\ C \cup \{ll' | l\mathtt{x} \in D, l' \in C'\},\\ \{l\mathtt{y} | l\mathtt{y} \in D\} \cup \{ll'\mathtt{y} | l\mathtt{x} \in D, l'\mathtt{y} \in D'\})\end{aligned}$$

6. Let $Z$ be countable. If $Y$ contains $\emptyset$ and a set $\geqslant 2^{\aleph_0}$, then every strategy into $\Gamma$ wrt $Z$ is $[M]$ for some term $\Gamma \vdash M$ in $\mathcal{L}(Y, Z)$.

□

(1)–(5) give us a compositional description of $[-]$, i.e. an adequate denotational semantics. However, we lack a characterization of those strategies definable using only countable choice.

# 3 Arenas and Pointer Games

## 3.1 Arenas and Strategies

The remainder of the paper uses the *pointer games* of [HO00]. We recapitulate below the basic theory of these games, omitting, for simplicity, the various constraints of innocence, visibility and bracketing (although the latter two can easily be incorporated into our model). Because of this liberality, our model includes general references [AHM98] and control operators [Lai98], though we will not treat these at the level of the language.

The formulation of sum types in [AHM98] is not quite suitable for us, because it needs both players to obey the bracketing condition. Instead, we follow the formulation in [Leva], where it is Player who moves first (as in Sect. 2.3).

**Definition 5** An (unlabelled) *arena* $R$ is a countable forest, i.e. a countable set $\mathsf{tok}\ R$ of *tokens* together with a function $\mathsf{enabler} : R \longrightarrow \{*\} + R$ such that, for every token $r$, the sequence

$$r = r_0 \overset{\mathsf{enabler}}{\longmapsto} r_1 \overset{\mathsf{enabler}}{\longmapsto} \cdots$$

eventually reaches $*$. We write $\mathsf{rt}\ R$ for the set of tokens enabled by $*$. □

An arena $R$ determines a game as follows. Play alternates between Player and Opponent, with Player moving first. In each move, a token $r \in \mathsf{tok}\ R$ is played. Player moves by *either* playing a root $r \in \mathsf{rt}\ R$, *or* pointing to a previous Opponent-move $m$ and playing a successor of the token played in $m$. Opponent moves by pointing to a previous Player-move $m$ and playing a successor of the token played in $m$. An arena $R$ together with an I/O signature $Z = \{I_o\}_{o \in O}$ determines a variation of this game: Player can opt to play some $o \in O$ instead of a token, and Opponent must then play some $i \in I_o$. Here is a formal description.

**Definition 6** 1. A *justified sequence* $s$ in an arena $R$ consists of
   - a finite or infinite sequence $s_0, s_1, \ldots$ in $\mathsf{tok}\ R + O + \sum_{o \in O} I_o$ (we classify $i \in \mathsf{dom}\ s$ as an *arena move*, an *o-output move* or an *o-input move*), such that a move is an $o$-input move iff it is the successor of an output move playing $o$
   - a function justifier mapping each arena move $m$ to an earlier arena move or $*$, called the *justification pointer* from $m$, such that $\mathsf{enabler} s_m$ is $s_{\mathsf{justifier}(m)}$ (or $*$ if $\mathsf{justifier} m = *$).
2. A *play* is a justified sequence $s$ such that
   - if $i$ is an arena move, then justifier $i$ is even if $i$ is odd, and odd or $*$ if $o$ is even
   - if $i$ is an output move, then $i$ is even.
3. A move $i \in \mathsf{dom}\ s$ is a *Player-move* or an *Opponent-move* according as $i$ is even or odd.
4. A finite play *awaits Player* or *awaits Opponent* according as its length is even or odd. In the latter case, it *awaits o-input* or *awaits arena-Opponent* according as its last move is an $o$-output move or an arena move.
5. An *nondeterministic infinite trace* (NIT) *strategy* $sigma$ for an arena $R$ consists of
   - a set $A$ of Opponent-awaiting plays (the *finite traces*)
   - a set $B$ of divergences (the *divergences*)

– a set $C$ of infinite plays (the *infinite traces*)

such that if $s$ is in $A$, $B$ or $C$, then every Opponent-awaiting prefix is in $A$. We write $R \xrightarrow{\sigma}$ to say that $\sigma$ is a strategy on $R$. We write $\mathsf{strat}$ for the set of strategies on $R$.

6. We define $\bigcup_{i\in I} \sigma_i$ and $\mathsf{input}^o_{i\in I_o}\ \sigma_i$ and *deterministic* strategies and *hiding* as in Def. 3.

□

The following cartesian category of arenas is crude but useful.

**Definition 7** 1. A *token-change* from arena $R$ to arena $S$ is a function $\mathsf{tok}\ S \xrightarrow{f} \mathsf{tok}\ R$, such that, if $b \in \mathsf{rt}\ S$, then $fb \in \mathsf{rt}\ R$ and $f$ restricts to an arena isomorphism from $S{\restriction_b}$ (the arena of tokens in $S$ strictly below $b$) to $R{\restriction_{fb}}$.

2. Given a token-change $R \xrightarrow{f} S$, and a strategy $\sigma$ on $S$, we define a strategy $f..\sigma$ on $R$. Its finite traces, divergences and infinite traces are obtained by applying $f$ to each token of those of $\sigma$.
3. We write $\mathbf{TokCh}$ for the category of arenas and token-changes. Finite products are given by disjoint union, which we write as $\uplus$.

□

### 3.2 Categorical Structure

In this section, we define a category $\mathcal{G}$, whose objects are arenas. It (with determinism and bracketing constraints) is called the "thread-independence" category in [AHM98].

**Definition 8** 1. For arenas $R$ and $S$, we define $\mathcal{G}(R,S)$ to be $\prod_{b\in\mathsf{rt}\ S} \mathsf{strat}\ (R \uplus S{\restriction_b})$.

2. For arena $R$ and $b \in \mathsf{rt}\ R$, we define $\mathsf{id}_{R,b}$ to be the deterministic strategy on $R \uplus R{\restriction_b}$ with no divergences, and whose finite/infinite traces are all plays in which Player initially plays $* \curvearrowleft \mathsf{inl}\ b$, and responds to

$$\begin{aligned} 0 \curvearrowleft \mathsf{inl}\ b \ \text{ with } \ * \curvearrowleft \mathsf{inr}\ b \\ n+1 \curvearrowleft \mathsf{inl}\ b \ \text{ with } \ n \curvearrowleft \mathsf{inr}\ b \\ n+1 \curvearrowleft \mathsf{inr}\ b \ \text{ with } \ n \curvearrowleft \mathsf{inl}\ b \end{aligned}$$

We then define $\mathsf{id}_R \in \mathcal{G}(R,R)$ to map $b \in \mathsf{rt}\ R$ to $\mathsf{id}_{R,b}$.

□

For arenas $R, S, T$, we define a map

$$\mathcal{G}(R,S) \times \mathsf{strat}\ (S \uplus T) \longrightarrow \mathsf{strat}\ (R \uplus T)$$

as follows.

**Definition 9** 1. An *interaction pre-sequence* on $R, S, T$ consists of a justified sequence on $R \uplus S \uplus T$—we write $\mathsf{threads}\ s$ for the set $\{*\} \cup \{m | s_m \in \mathsf{rt}\ S\}$—together with a function mapping each rootmove in $R$ to an earlier rootmove in $S$, and each output move to an element of $\mathsf{threads}\ s$. (These are called *thread-pointers*.)

2. Given an interaction pre-sequence $s$ on $R, S, T$,
   - its *outer thread* consists of all moves in $R$ and $T$, and all output and input moves
   - its $*$-*inner thread* consists of all moves in $S$ and $T$, and all output moves thread-pointing to $*$ and subsequent input moves
   - its $q$-inner thread (where $q$ is a rootmove in $S$) consists of all $R$ moves descended from a rootmove threadpointing to $q$, all $S$ moves *strictly* descended from $q$, and all output move thread-pointing to $*$ and subsequent input moves.

   $s$ is an *interaction sequence* when all these justified sequences are plays.
3. Let $s$ be an interaction sequence, and let $q \in \{\mathsf{outer}\} + \mathsf{threads}\ s$ (we say that $q$ is a *thread-index*). Then $q$ is *live* in $s$ when the $q$-thread awaits Opponent, if $q = \mathsf{outer}$, and awaits Player, if $q \in \mathsf{threads}\ s$.

□

**Proposition 5** Let $s$ be a finite interaction sequence on $R, S, T$.

- $s$ has precisely one live thread-index, call it $q$.
- If $sm$ is an interaction sequence, then $m$ is in the $q$-thread of $sm$, and so $q$ is not live in $sm$.
- If $s$ has $q$-thread $t$, and $tm$ is a play, then $sm$ is an interaction sequence.

If $s$ is an infinite interaction sequence, then no thread-index is live. □

**Definition 10** 1. Let $R, S, T$ be arenas, let $\sigma \in \mathcal{G}(R, S)$ and let $\tau \in \mathsf{strat}\ (S \uplus T)$. We define $\sigma \backslash \tau \in \mathsf{strat}\ (R \uplus T)$ to have

   **finite traces** the outer thread of every outer-Opponent awaiting play $s$ whose $q$-inner thread, for every $q \in \mathsf{threads}\ s$, is a finite trace of $q(\sigma, \tau)$

   **divergences (1)** the outer thread of every $l$-inner-Player awaiting play $s$ whose $l$-inner thread is a divergence of $l(\sigma, \tau)$ and whose $q$-inner thread, for every $q \in \mathsf{threads}\ s \setminus \{l\}$, is a finite trace of $q(\sigma, \tau)$

   **divergences (2)** the outer thread of every infinite play $s$ whose $q$-inner thread, for every $q \in \mathsf{threads}\ s$, is a finite trace or infinite trace of $q(\sigma, \tau)$, and whose outer thread awaits Player

   **infinite traces** the outer thread of every outer-infinite play $s$ whose $q$-inner thread, for every $q \in \mathsf{threads}\ s$, is a finite trace or infinite trace of $q(\sigma, \tau)$, and whose outer thread is infinite.
2. Given $\mathcal{G}$-morphisms $R \xrightarrow{\sigma} S$ and $S \xrightarrow{\tau} T$, we define the composite $R \xrightarrow{\sigma;\tau} T$ at $b \in \mathsf{rt}\ T$ to be $\sigma \backslash \tau_b$.
3. Given $\mathcal{G}$-morphism $R \xrightarrow{\sigma} S$ and $S \xrightarrow{\tau}$, we define the composite $R \xrightarrow{\sigma;\tau}$ to be $\sigma \backslash \tau$ (taking $T$ to be the empty arena).

□

**Proposition 6** Def. 10(2) satisfies associativity and identity laws, making $\mathcal{G}$ a category. Def. 10(3) satisfies associativity and left-identity laws, making $\mathsf{strat}$ a left $\mathcal{G}$-module, i.e. a functor $\mathcal{G}^{\mathrm{op}} \longrightarrow \mathbf{Set}$. □

We define an identity-on-objects functor $\mathcal{F} : \mathbf{TokCh} \longrightarrow \mathcal{G}$, by token-changing copycat strategies. Then all compositions of the form $R \xrightarrow{\mathcal{F}f} S \xrightarrow{\sigma} T$ or $R \xrightarrow{\mathcal{F}f} S \xrightarrow{\sigma}$ or $R \xrightarrow{\sigma} S \xrightarrow{\mathcal{F}f} T$ are trivial, because they just token-change along $f$. It is immediate that $\mathcal{G}$ has finite products given by $\uplus$, and $\mathcal{F}$ preserves finite products on the nose.

We recover $\backslash$ from the categorical structure:

**Proposition 7** If $R \xrightarrow{\sigma} S$ and $R \uplus T \xrightarrow{\tau}$, then $\sigma \backslash \tau = (\sigma \times T); \tau$ □

By analogy with Prop. 2, we have

**Proposition 8**
- The operations $\backslash$ and token-changing and $\mathsf{input}^o$ all preserve determinism, and $\mathsf{id}_{R,b}$ is deterministic.
- Given signatures $Z$ and $Z'$, the hiding of

$$\begin{array}{rcl} \mathsf{id}_{R,b} & \text{is} & \mathsf{id}_{R,b} \\ f..\sigma & \text{is} & f..(\sigma \upharpoonright Z) \\ \sigma \backslash \tau & \text{is} & (\sigma \upharpoonright Z) \backslash (\tau \upharpoonright Z) \\ \bigcup_{i \in I} \sigma_i & \text{is} & \bigcup_{i \in I} (\sigma_i \upharpoonright Z) \\ \mathsf{input}^o_{i \in I_o}\ \sigma_i & \text{is} & \begin{cases} \mathsf{input}^o_{i \in I_o}\ (\sigma_i \upharpoonright Z) & \text{if } o \in Z \\ \bigcup_{i \in I_o} (\sigma_i \upharpoonright Z) & \text{if } o \in Z' \end{cases} \end{array}$$

where $\sigma$ and $\tau$ and all $\sigma_i$ are strategies wrt $Z + Z'$.

□

In order to give the semantics of `inl` and `inr`, we require the following, whose direct description we omit.

**Definition 11** If $b \in \mathsf{rt}\ S$ and $R \xrightarrow{\sigma} S{\upharpoonright_b}$, write $R \uplus S \xrightarrow{b \ltimes \sigma}$ for $\sigma \backslash (f..\mathsf{id}_{S,b})$, writing $S{\upharpoonright_b} \uplus S \xrightarrow{f} S \uplus S{\upharpoonright_b}$ for the obvious token-change. □

# 4 Call-By-Name FPC

## 4.1 Operational Semantics

Again, let $Y$ be an erratic signature and let $Z$ be an I/O signature. We define a language $\mathcal{L}^{\mathsf{FPC}}(Y, Z)$ as follows. The types are given by

$$A ::= \quad A + A \mid 0 \mid A \times A \mid 1 \mid A \to A \mid X \mid \mu X.A$$

0 is, in effect, the sole type of the language of Sect. 2.4. The terms are given (omitting the constructs for 0 and 1) by

$$\begin{array}{rcl} M ::= & \mathtt{x} \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \mid \pi M \mid \pi' M \\ & \mid \lambda \mathtt{x}.M \mid MM \mid \mathtt{fold}\ M \mid \mathtt{unfold}\ M \\ & \mid \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} \mid (M, M) \end{array}$$

where pm stands for "pattern-match". We define $\Gamma \vdash M : B$ in the standard way.

We give CK-machine semantics [FF86] in Fig. 1. During evaluation of $\Gamma \vdash N : C$, a configuration consists of a term $\Gamma \vdash M : B$ and a stack $K$ of contexts. We introduce a typing judgement for the stack $\Gamma | B \vdash^{\mathsf{k}} K : C$, inductively defined in Fig. 2. For each context $\Gamma$ and stack $C$, Fig. 1 gives a terminable BLTS wrt $Z$ called $\mathcal{L}^{\mathsf{FPC}}(Y, Z, \Gamma, C)$.

| **Initial Configuration** to execute $\Gamma \vdash N : C$ | | | | |
|---|---|---|---|---|
| $\Gamma$ $N$ | $C$ | $\mathtt{nil}$ | $C$ | |
| **Transitions** | | | | |
| $\Gamma$ $\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\}$ | $B$ | $K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $M$ | $A + A'$ | $\mathtt{pm}\ [\cdot]\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} :: K$ | $C$ | |
| $\Gamma$ $\mathtt{inl}\ P$ | $A + A'$ | $\mathtt{pm}\ [\cdot]\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} :: K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $N[P/\mathtt{x}]$ | $B$ | $K$ | $C$ | |
| $\Gamma$ $\pi M$ | $B$ | $K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $M$ | $B \times B'$ | $\pi[\cdot] :: K$ | $C$ | |
| $\Gamma$ $(N, N')$ | $B \times B'$ | $\pi[\cdot] :: K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $N$ | $B$ | $K$ | $C$ | |
| $\Gamma$ $MN$ | $B$ | $K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $M$ | $A \to B$ | $[\cdot]N :: K$ | $C$ | |
| $\Gamma$ $\lambda\mathtt{x}.P$ | $A \to B$ | $[\cdot]N :: K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $P[N/\mathtt{x}]$ | $B$ | $K$ | $C$ | |
| $\Gamma$ $\mathtt{unfold}\ M$ | $B[\mu X.B/X]$ | $K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $M$ | $\mu X.B$ | $\mathtt{unfold}\ [\cdot] :: K$ | $C$ | |
| $\Gamma$ $\mathtt{fold}\ N$ | $\mu X.B$ | $\mathtt{unfold}\ [\cdot] :: K$ | $C$ | $\rightsquigarrow$ |
| $\Gamma$ $N$ | $B[\mu X.B/X]$ | $K$ | $C$ | |
| $\Gamma$ $\mathtt{choose}^{h}_{p \in P_h}\ M_p$ | $B$ | $K$ | $C$ | $\rightsquigarrow \quad (\hat{p} \in P_h)$ |
| $\Gamma$ $M_{\hat{p}}$ | $B$ | $K$ | $C$ | |
| $\Gamma$ $\mathtt{input}^{o}_{i \in I_o}\ M_i$ | $B$ | $K$ | $C$ | $: \hat{\imath} = \quad (\hat{\imath} \in I)$ |
| $\Gamma$ $M_{\hat{\imath}}$ | $B$ | $K$ | $C$ | |
| **Terminal Configurations** | | | | |
| $\Gamma$ $\lambda\mathtt{x}.P$ | $A \to B$ | $\mathtt{nil}$ | $A \to B$ | |
| $\Gamma$ $(N, N')$ | $B \times B'$ | $\mathtt{nil}$ | $B \times B'$ | |
| $\Gamma$ $\mathtt{inl}\ M$ | $A + A'$ | $\mathtt{nil}$ | $A + A'$ | |
| $\Gamma$ $\mathtt{fold}\ M$ | $\mu X.B$ | $\mathtt{nil}$ | $\mu X.B$ | |
| $\Gamma$ $\mathtt{x}$ | $B$ | $K$ | $C$ | |

**Fig. 1.** CK-machine semantics for call-by-name FPC

### 4.2 Denotational Semantics

**Definition 12** 1. A Q/A-*labelled arena* is an arena $(R, \mathsf{enabler})$, with every token classified as *question* or *answer*, where no answer enables an answer. It is Q-*rooted* when, moreover, every root is a question.

$$\frac{}{\Gamma|C \vdash^{\mathsf{k}} \mathtt{nil} : C} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{k}} N : B \quad \Gamma, \mathtt{x} : A \vdash N' : B \quad \Gamma|B \vdash^{\mathsf{k}} K : C}{\Gamma|A + A' \vdash^{\mathsf{k}} \mathtt{pm}\,[\cdot]\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} :: K : C}$$

$$\frac{\Gamma \vdash N : B \quad \Gamma|B \vdash^{\mathsf{k}} K : C}{A \to \Gamma|B \vdash^{\mathsf{k}} [\cdot]N :: K : C} \qquad \frac{\Gamma|B[\mu X.B/X] \vdash^{\mathsf{k}} K : C}{\Gamma|\mu X.B \vdash^{\mathsf{k}} \mathtt{unfold}\,[\cdot] :: K : C} \qquad \frac{\Gamma|B' \vdash^{\mathsf{k}} K : C}{\Gamma|B \times B' \vdash^{\mathsf{k}} \pi[\cdot] :: K : C}$$

**Fig. 2.** Stack syntax for call-by-name FPC

2. For a countable family of Q/A-labelled arenas $\{R_i\}_{i\in I}$, we write $\mathsf{pt}^{\mathsf{Q}}_{i\in I} R_i$ for the labelled arena with $I$ roots, each a question, and a copy of $R_i$ placed below the $i$th root. Similarly $\mathsf{pt}^{\mathsf{A}}_{i\in I} R_i$, provided that each $R_i$ is Q-rooted.
3. Let $R$ and $S$ be Q/A-labelled arenas. We say that $R \sqsubseteq S$ when for every $r \in \mathsf{tok}\ R$, both $r$ and all its ancestors are tokens of $S$, with the same labelling.
4. We write $\mathcal{E}$ for the (non-small) cpo of countable families of Q/A-labelled arenas. $\{R_i\}_{i\in I} \sqsubseteq \{S_j\}_{j\in J}$ when for every $i \in I$, we have $j \in J$ and $R_i \sqsubseteq S_i$.

□

A type with $n$ free identifiers denotes a continuous function from $\mathcal{E}^n$ to $\mathcal{E}$, with type recursion interpreted as least fixpoint. If, in a given type environment $\rho \in \mathcal{E}^n$, type $A$ denotes $\{R_i\}_{i\in I}$ and type $B$ denotes $\{S_j\}_{j\in J}$, then, at $\rho$,

- $A \times B$ denotes the combined family indexed by $I + J$
- $A \to B$ denotes $\{\mathsf{pt}^{\mathsf{Q}}_{i\in I} R_i \uplus S_j\}_{j\in J}$
- $A + B$ denotes $\{\mathsf{pt}^{\mathsf{A}}\{\mathsf{pt}^{\mathsf{Q}}_{i\in I} R_i, \mathsf{pt}^{\mathsf{Q}}_{j\in J} S_j\}\}$.

Semantics of judgements:

- A context $\Gamma = \mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$, where $A_k$ denotes $\{R_{ki}\}_{i\in I_k}$, denotes the labelled arena $\mathsf{pt}^{\mathsf{Q}}_{i\in I_0} R_{0i} \uplus \cdots \uplus \mathsf{pt}^{\mathsf{Q}}_{i\in I_{n-1}} R_{(n-1)i}$.
- If the context $\Gamma$ denotes $R$, and the type $B$ denotes $\{S_j\}_{j\in J}$, then a term $\Gamma \vdash M : B$ and a configuration $\Gamma, M, A, K, B$ both denote an element of $\prod_{j\in J} \mathsf{strat}\,(R \uplus S_j)$
- If the context $\Gamma$ denotes $R$ and $A$ denotes $\{S_j\}_{j\in J}$ and the type $B$ denotes $\{T_k\}_{k\in K}$, then a stack $\Gamma|B \vdash^{\mathsf{k}} K : C$ denotes an element of $\prod_{k\in K} \sum_{j\in J} \mathcal{G}(R \uplus T_k, S_j)$.

Semantics of terms (in outline)

- $\mathtt{choose}^h$ and $\mathtt{input}^o$ are interpreted by $\bigcup$ and $\mathsf{input}^o$.
- Identifiers and `nil` denote token-changed id strategies.
- The operations of projection, pairing, $\lambda$, fold, unfold and stacking application, projection and fold contexts are interpreted by token-changing.
- `inl`/`inr` are interpreted using $\ltimes$ and token-changing.
- The operations of application, pattern-match, stacking a pattern-match context and forming a configuration are interpreted using $\setminus$ and token-changing.

**Definition 13** For a configuration $d = \Gamma, M, B, K, C$, where $\Gamma$ denotes $R$ and $C$ denotes $\{S_j\}_{j\in J}$, we write $[\![d]\!]$ for the element of $\prod_{j\in J} \mathsf{strat}\,(R \uplus S_j)$ that maps $j$ to the strategy on $S_j$ containing

- all finite traces/divergences/infinite traces of $[d]$
- all finite traces/divergences/infinite traces of the form $st$, where $sT$ is a terminating trace of $[d]$, and $t$ is a finite trace/divergence/infinite trace of $[\![T]\!]j$.

□

Using Prop. 8(8) and Prop. 7, we prove

**Proposition 9** 1. If the erratic signature $Y$ is empty, then the denotation of every term, stack and configuration $d$ is deterministic, and so is $[\![d]\!]$.
2. [soundness] For any configuration $d$, we have $[\![d]\!] =$
   - $[\![M]\!]$ if $d = \Gamma, M, C, \mathtt{nil}, C$
   - $\bigcup_{d\rightsquigarrow d'} [\![d']\!]$, if $d'$ is silent
   - $\mathsf{input}^{o}_{i\in I_o}\ [\![d : i]\!]$ if $d$ is an $o$-state

□

Finally, we want to prove

**Proposition 10 (adequacy)** $[\![d]\!] = [\![d]\!]$, for every configuration $d$ in $\mathcal{L}^{\mathsf{FPC}}(Y, Z)$. □

Prop. 9(2) immediately gives us a weak version:

**Lemma 2** Let $d = \Gamma, M, B, K, C$, where $[\![C]\!] = \{S_j\}_{j\in J}$. Suppose $j \in J$.

1. If $s(T, \mathtt{nil}\ )$ is a terminating trace of $[d]$ and $t$ is a finite trace (divergence, infinite trace) of $[\![T]\!]j$, then $st$ is a finite trace (divergence, infinite trace) of $[\![d]\!]j$.
2. Every finite trace of $[\![d]\!]j$ is a finite trace of $[\![d]\!]j$.
3. Every finite trace (divergence, infinite trace) of $[\![d]\!]j$ is either a finite trace (divergence, infinite trace) of $[\![d]\!]j$ or an extension of a divergence of $[d]$.

□

We next define the *unhiding transform* from $\mathcal{L}^{\mathsf{FPC}}(Y, Z)$ to $\mathcal{L}^{\mathsf{FPC}}(\{\}, Z + (Y + \{\checkmark\}))$. The translation on terms and stacks is defined in Fig. 3. For a configuration $d = M, B, K, C$, define $\overline{d}$ to be the configuration $\overline{M}, B, \overline{K}, C$.

**Lemma 3** 1. If $\Gamma, \mathtt{x} : A \vdash M : B$ and $\Gamma \vdash N : A$ then $\overline{M[N/\mathtt{x}]} = \overline{M}[\overline{N}/\mathtt{x}]$.
2. Let $d = M, B, K, C$. If $M$ is not $\mathtt{choose}$ or $\mathtt{input}$, then either $d$ and $d'$ are both terminal, or $d \rightsquigarrow d'$ for unique $d'$, and $\overline{d} \rightsquigarrow \checkmark.\overline{d'}$.
3. $[\overline{d}]$ has no divergences.
4. If $[d] = (A, B, C, D)$ then $[\overline{d}] \upharpoonright Z = (A, B, C, \{s\overline{T} | sT \in D\})$

□

Now $[\![\overline{d}]\!]$ and $[\![\overline{d}]\!]$ are deterministic (Prop. 9(1)) and have the same finite traces (Lemma 2(2)–(3) and Lemma 3(3)). So $[\![\overline{d}]\!] = [\![\overline{d}]\!]$. Prop. 8(8) tells us the following.

**Lemma 4** 1. If $P$ is a term or stack, then $[\![\overline{P}]\!] \upharpoonright Z = [\![P]\!]$.
2. If $d$ is a configuration, then $[\![\overline{d}]\!] \upharpoonright Z = [\![d]\!]$, so (by Lemma 3(4)) $[\![\overline{d}]\!] \upharpoonright Z = [\![d]\!]$.

□

Hence $[\![d]\!] = [\![\overline{d}]\!] \upharpoonright Z = [\![\overline{d}]\!] \upharpoonright Z = [\![d]\!]$, as required.

| $\Gamma \vdash M : B$ | $\Gamma \vdash \overline{M} : B$ |
|---|---|
| $\mathtt{x}$ | $\mathtt{x}$ |
| $\lambda \mathtt{x}.M$ | $\lambda \mathtt{x}.\checkmark.\overline{M}$ |
| $MN$ | $(\checkmark.\overline{M})\overline{N}$ |
| $(M, M')$ | $(\overline{M}, \overline{M'})$ |
| $\pi M$ | $\pi\checkmark.\overline{M}$ |
| $\mathtt{inl}\ M$ | $\mathtt{inl}\ \overline{M}$ |
| $\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\}$ | $\mathtt{pm}\ \checkmark.\overline{M}\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.\checkmark.\overline{N}, \mathtt{inr}\ \mathtt{x}.\checkmark.\overline{N'}\}$ |
| $\mathtt{fold}\ M$ | $\mathtt{fold}\ \checkmark.\overline{M}$ |
| $\mathtt{unfold}\ M$ | $\mathtt{unfold}\ \checkmark.\overline{M}$ |
| $\mathtt{choose}^{h}_{p \in P_h}\ M_p$ | $\mathtt{input}^{h}_{p \in P_h}\ M_p$ |
| $\mathtt{input}^{o}_{i \in I_o}\ M_i$ | $\mathtt{input}^{o}_{i \in I_o}\ M_i$ |
| $B \vdash^{\mathsf{k}} K : C$ | $B \vdash^{\mathsf{k}} \overline{K} : C$ |
| $\mathtt{nil}$ | $\mathtt{nil}$ |
| $[\cdot]N :: K$ | $[\cdot]\overline{N} :: \overline{K}$ |
| $\pi[\cdot] :: K$ | $\pi[\cdot] :: \overline{K}$ |
| $\mathtt{pm}\ [\cdot]\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} :: K$ | $\mathtt{pm}\ [\cdot]\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.\checkmark.\overline{N}, \mathtt{inr}\ \mathtt{x}.\checkmark.\overline{N'}\} :: \overline{K}$ |
| $\mathtt{unfold}\ [\cdot] :: K$ | $\mathtt{unfold}\ [\cdot] :: \overline{K}$ |

**Fig. 3.** The Unhiding Transform

## References


[AHM98] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proc., 13th Ann. IEEE Symp. on Logic in Comp. Sci.*, 1998.

[Bro02] S. Brookes. The essence of Parallel Algol. *Information and Computation*, 179, 2002.

[Esc98] M.H. Escardó. A metric model of PCF. unpublished research note, 1998.

[FF86] M. Felleisen and D. Friedman. Control operators, the SECD-machine, and the $\lambda$-calculus. In M. Wirsing, editor, *Formal Description of Prog. Concepts*. North-Holland, 1986.

[HM99] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proc., 14th Ann. IEEE Symp. on Logic in Comp. Sci.*, 1999.

[HO00] M. Hyland and L. Ong. On full abstraction for PCF: I, II, and III. *Inf. and Comp.*, 163(2), 2000.

[Lai98] J. Laird. *A Semantic Analysis of Control*. PhD thesis, University of Edinburgh, 1998.

[Leva] P. B. Levy. Adjunction models for call-by-push-value with stacks. to appear in Theory and Applications of Categories.

[Levb] P. B. Levy. Infinite trace semantics. Proc., 2nd APPSEM II Workshop, Tallinn, April, 2004 `www.cs.ioc.ee/appsem04/accepted.html`.

[Lev04] P. B. Levy. *Call-By-Push-Value*. Semantic Structures in Computation. Kluwer, 2004.

[McC96] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. PhD thesis, University of London, 1996.

[Mog91] E. Moggi. Notions of computation and monads. *Inf. and Comp.*, 93, 1991.

[Plo83] G. Plotkin. Domains. prepared by Y. Kashiwagi, H. Kondoh and T. Hagino., 1983.

[PP02] G. Plotkin and J. Power. Notions of computation determine monads. In *Proc., Foundations of Software Sci. and Comp. Struct., 2002*, volume 2303. LNCS, 2002.

[Ros98] A. W. Roscoe. *Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[Ros04] A. W. Roscoe. Seeing beyond divergence. presented at BCS FACS meeting "25 Years of CSP", July 2004.