Infinite Trace Equivalence

Paul Blain Levy [1]

University of Birmingham, U.K.

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.

Key words: nondeterminism, infinite traces, game semantics

1 Introduction

1.1 The Problem

Consider the following call-by-name [2] language of countably nondeterministic commands:

$$M ::= \quad \mathtt{x} \mid \mathtt{print}\ c.\ M \mid \mu\mathtt{x}.M \mid \mathtt{choose}\ n_{\in\mathbb{N}}.\ M_n$$

where c ranges over some alphabet $\mathcal{A}$. We define binary nondeterminism $M\ \mathtt{or}\ M'$ from countable in the evident way.

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 [15], "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

[1] Email: pbl@cs.bham.ac.uk

[2] Meaning that an identifier gets bound to an unevaluated term.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

trace equivalence. Some models of nondeterminism, such as the various powerdomains [15] and divergence semantics [18], identify programs that are not infinite trace equivalent, so they are too coarse. Others count the internal manipulations [2,4] 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 [8], although the technology of pointer games is needed only 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 [13].

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 $\{\checkmark\}$.

(i) 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 [15], all the CSP semantics in [17], and the game semantics of [6]. Divergence-least semantics cannot model infinite trace equivalence, by the following argument taken from [15]. (We abbreviate $\mathtt{print}\ \checkmark$ as $\checkmark$.)

$$\begin{array}{rl} \text{Put} & M \;=\; \bot \;\mathtt{or}\; \checkmark.\checkmark.\bot \\ & M' \;=\; \bot \;\mathtt{or}\; \checkmark.\bot \;\mathtt{or}\; \checkmark.\checkmark.\bot \\ \text{Then} & M \;=\; \bot \;\mathtt{or}\; \bot \;\mathtt{or}\; \checkmark.\checkmark.\bot \;\leqslant\; M' \\ & M \;=\; \bot \;\mathtt{or}\; \checkmark.\checkmark.\bot \;\mathtt{or}\; \checkmark.\checkmark.\bot \;\geqslant\; M' \end{array}$$

Hence $M = M'$, contradicting infinite trace equivalence. This argument uses only binary nondeterminism.

(ii) 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 [15], all the CSP semantics in [17], the semantics using infinite traces in [2], and divergence semantics [18]. In general, well-pointed semantics are 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 [3] involving $\mathtt{x}$ demonstrate:

[3] discovered by A. W. Roscoe in 1989 [personal communication], and independently in [10].

$$N = (\texttt{choose}\ n.\ \checkmark^n.\ \mu\texttt{z.z})\ \texttt{or}\ \texttt{x}$$
$$N' = (\texttt{choose}\ n.\ \checkmark^n.\ \mu\texttt{z.z})\ \texttt{or}\ \checkmark.\texttt{x}$$

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

(Lest the reader think unbounded nondeterminism is to blame, suppose we allow only binary nondeterminism, but put $\mathbb{N}$-indexed commands into the language. Then define $\texttt{choose}^\perp\ n_{\in\mathbb{N}}.M_n$ to be $(\mu\texttt{f}\lambda n.(M_n\ \texttt{or}\ \texttt{f}(n+1)))0$, which either executes some M_n or diverges [15]. Using this instead of $\texttt{choose}\ n_{\in\mathbb{N}}$, the same problem arises.)

Naively, N and N' can be distinguished by saying that N' is able to print a tick and then force (i.e. execute) 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 [11]) "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 [6], where strategy sets are quotiented by the Egli-Milner preorder 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.

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

1.4 Related Work: Dataflow Networks

An infinite trace model for *dataflow networks*—including feedback, but not recursion—was presented in [9], and shown fully abstract. In the terminology of [7], it forms a *cartesian-centre traced SMC*. It is shown in [7] that such a category can interpret recursion in a certain sense—provided it is "centrally closed", which Jonsson's model (like its finite trace variant) is not.

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.

2 First-Order Language

2.1 Erratic Choice and Omni-Errors

We wish to allow choice operators of arbitrary arity, including empty. But a language containing a command "choose an element of the empty set" has no valid implementation. We skirt this problem as follows.

Definition 2.1 An *erratic signature* Y is a family of sets $\{P_h\}_{h\in H}$, together with a set U. It is *deadlock-free* when 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

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

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). In this paper, we study only deadlock-free signatures.

A program in this language can behave in 3 ways:

(i) print finitely many characters then diverge

(ii) print infinitely many characters

(iii) print finitely many characters then raise an omni-error.

For a closed term M, let us write $[M]_U \subset \mathcal{A}^* + \mathcal{A}^\omega + \mathcal{A}^* \times U$ for the set of possible behaviours. We define *infinite trace equivalence* to be the kernel of $[-]_U$. Clearly

$$[M]_U = [M]_{\mathsf{inf}} + [M]_{\mathsf{fin}} \times U$$

where $[M]_{\mathsf{inf}} \subset \mathcal{A}^* + \mathcal{A}^\omega$ is the range of behaviours of type (i)–(ii), and $[M]_{\mathsf{fin}} \subset \mathcal{A}^*$ is the set of finite traces of M. Let us write $[M]$ for the pair $([M]_{\mathsf{fin}}, [M]_{\mathsf{inf}})$.

Proposition 2.2 (i) For some deadlock-free signatures and alphabets, infinite trace equivalence is strictly finer than the kernel of $[-]_{\mathsf{inf}}$.

(ii) For all deadlock-free signatures and alphabets, infinite trace equivalence is the kernel of $[-]$.

□

Proof. (i) Consider $\checkmark.\mathtt{choose}^h$ and $\mathtt{choose}^h$, where P_h is empty.
(ii) By deadlock-freeness, every finite path extends to a path that is either infinite or ends in an omni-error. □

Prop. 2.2(ii) legitimates leaving omni-errors out of a semantics of infinite trace equivalence (for deadlock-free signatures), provided we include the finite traces.

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 [14,16], to model requested input. Note that this includes as special cases the monads designated in [14] 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 [4] for $\mathtt{input}^c_{i \in 1}\ M$.

2.3 Bi-Labelled Transition Systems

To describe the behaviour of a system using requested input with signature Z, an LTS (i.e. a coalgebra for the endofunctor $\mathcal{P}(A \times -)$, for some fixed set A of actions) is not really suitable. If we allow both outputs and inputs to be actions, we need additional alternation and receptivity-to-input conditions. And if we define an action to be a pair (o, i), we do not deal with the case of an output whose input never arrives (or, indeed, whose input set is empty).

Instead, we use the following concept (abstractly, coalgebra for the endofunctor $\mathcal{P} + R_Z$ on $\mathbf{Set}$).

Definition 2.3 (BLTS)

(i) A *bi-labelled transition system* (BLTS) $\mathcal{L}$, wrt an I/O signature $Z = \{I_o\}_{o \in O}$ and set of omni-errors U, consists of

[4] 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_{sil} and S_o for the set of silent states and of o-states, respectively
- a relation $\rightsquigarrow$ from S_{sil} to S, and, for each $o \in O$, a function $S_o \times I_o \xrightarrow{\cdot\,:\,} S$.

It is *deadlock-free* if either every silent state has at least one successor, or U is nonempty.

(ii) A *terminable BLTS* is the same, except that there is a third kind of state: *terminal*.

(iii) A terminable BLTS, is *deterministic* when U is empty and each silent state has precisely one successor.

□

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

Definition 2.4 (strategies) Let Z be an I/O signature, and let V be a set.

(i) 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.

(ii) 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^{NIT} is a strong monad on $\mathbf{Set}$.

(iii) A Player-awaiting or infinite play is *consistent* with a strategy σ when every input-awaiting prefix is a finite trace of σ.

(iv) For $d \in V$, we define ηd (the monad's unit at d) to be the strategy

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

(v) 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

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

(vi) 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

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

(vii) 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.

(viii) Given a Player-awaiting (resp. infinite) play s wrt $Z+Z'$, we write $s \restriction Z$ for the Player-awaiting (resp. Player-awaiting or infinite) play obtained by removing moves in Z'.

(ix) Given a strategy $\sigma = (A, B, C, D)$ into V wrt $Z + Z'$, the *hiding* of σ, written $\sigma \restriction Z$, is the strategy wrt Z given by

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

□

Proposition 2.5 (i) The strategy ηz is deterministic, and input^o preserves determinism.

(ii) Given signatures Z and Z', the hiding of

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

where each σ_i is a strategy wrt $Z + Z'$.

□

Definition 2.6 (BLTS to strategies) Let Z be an I/O signature, and let $\mathcal{L}$ be a deadlock-free terminable BLTS wrt Z and U. Write V for its set of terminal states.

(i) 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.

(ii) Two states d and d' are *infinite trace equivalent* when $[d] = [d']$.

□

Proposition 2.7 For any state d in a deadlock-free terminable BLTS, $[d]$ is $\mathsf{input}^o_{i \in I_o}\ [d : i]$ or $\bigcup_{d \rightsquigarrow d'} [d']$ or ηd 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 $Y = (\{P_h\}_{h \in H}, U)$ 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 and U, 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

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

$\mathcal{L}(Y, Z, \Gamma)$ is deadlock-free iff Y is. These transition systems have the following properties.

Lemma 2.8 Suppose $\Gamma, \mathtt{x} \vdash M$ and $\Gamma \vdash N$. Suppose that M is not $\mathtt{x}$.

(i) 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 $Q = M'[N/\mathtt{x}]$.

(ii) 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 $i \in I_o$.

(iii) For each $\mathtt{y} \in \Gamma$, we have $M = \mathtt{y}$ iff $M[N/\mathtt{x}] = \mathtt{y}$.

□

The key result is that we can characterize $[-]$ in a compositional way:

Proposition 2.9 In $\mathcal{L}^{\mathsf{FPC}}(Y, Z)$, where Y is deadlock-free, we have

(i) $[\mathtt{x}] = \eta \mathtt{x}$

(ii) $[\mathtt{choose}^h_{p \in P_h}\ M_p] = \bigcup_{p \in P_h} [M_p]$

(iii) $[\mathtt{input}^o_{i \in I_o}\ M_i] = \mathsf{input}^o_{i \in I_o}\ [M_o]$

(iv) If $\Gamma, \mathtt{x} \vdash M$ then $[\mu \mathtt{x}.M] = \mu[M]$, where we define $\mu(A, B, C, D)$ to be

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

(v) 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

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

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

□

(i)–(v) give us a computationally adequate denotational semantics. However, we lack a characterization of those strategies definable using only countable choice.

3 Call-By-Name FPC

For an erratic signature $Y = (\{P_h\}_{h \in H}, U)$ and an I/O signature $Z = \{I_o\}_{o \in I}$, we define the higher-order language $\mathcal{L}^{\mathsf{FPC}}(Y, Z)$ to have types 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

$$\begin{aligned}
M ::= \quad & \mathtt{x} \mid \mathtt{inl}\, M \mid \mathtt{inr}\, M \mid \pi M \mid \pi' M\\
& \mid \lambda \mathtt{x}.M \mid MM \mid \mathtt{fold}\, M \mid \mathtt{unfold}\, M\\
& \quad \mid \mathtt{pm}\, M\, \mathtt{as}\, \{\mathtt{inl}\, \mathtt{x}.N, \mathtt{inr}\, \mathtt{x}.N'\} \mid (M, M)\\
& \mid \mathtt{choose}^h_{p \in P_h}\, M_p \mid \mathtt{input}^o_{i \in I_o}\, M_i
\end{aligned}$$

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

We give CK-machine semantics [5] in Fig. 1. As we work on a term, we keep a stack of contexts, similar to an evaluation context. We write $\Gamma | B \vdash^{\mathsf{k}} K : C$ to mean

that K is a stack that can accompany a term of type B in context Γ, in the course of evaluating a term of type C in context Γ. This judgement is defined inductively in Fig. 2.

We define a *configuration* inhabiting $\Gamma \vdash C$ to consist of a type B, a term $\Gamma \vdash M : B$ and a stack $\Gamma | B \vdash^{\mathsf{k}} K : C$. We write $\mathcal{L}^{\mathsf{FPC}}(Y, Z, \Gamma \vdash C)$ for the terminable BLTS wrt Z and Y whose states are the configurations inhabiting $\Gamma \vdash C$, and which is defined in Fig. 1. It is deadlock-free iff Y is. **Note** Formally, all terms are deemed to be explicitly typed, and these types are used in Fig. 1 (making the machine deterministic, other than for `choose` terms), although we have not written them in.

4 Pointer Games

4.1 Pointer Game On Arena

We obtain our model of CBN FPC by taking the standard game semantics of [13]—omitting for simplicity, the constraints of innocence, visibility and bracketing, although the latter two could easily be incorporated—and remove the nondeterminism constraint in the manner of Def. 2.4(ii). To make the semantics of sum types work smoothly (the formulation in [1] can only work with the bracketing condition), we make two superficial changes in the presentation:

- a type denotes a family of arenas, rather than a single arena
- a term denotes a family of Player-first strategies, rather than a single Opponent-first strategy.

We have to give the denotation of stacks, as well as terms, and this may appear to be pulled out of a hat: a categorical explanation can be found in [12].

We begin with the basic concept of pointer game semantics. Let R be a countable forest, or *arena*. R determines the following two-player game. Play alternates between Player and Opponent, with Player moving first. In each move, an element of R is played. Player moves by *either* stating a root $r \in \mathsf{rt}\ R$, *or* pointing to a previous Opponent-move m and stating a child of the element played in m. Opponent moves by pointing to a previous Player-move m and stating a child of the element played in m.

This game, called the *pointer game* on R, may seem mysterious. In what sense does a higher-order program play such a game? A concrete explanation is given in [11], using a language and a style of operational semantics that are more explicit about interaction between parts of programs (see also [3]). Since all this is orthogonal to the nondeterminism which is the subject of this paper, we omit it.

An I/O signature Z determines a variation on this game: Player can opt to play some $o \in O$ instead of an R-element, and Opponent must then play some $i \in I_o$. We call this the pointer game on R wrt Z. Our first step is to formalize a play of this game, including all the pointers between moves.

Definition 4.1 (i) A *justified sequence* s in an arena R wrt I/O signature Z con-

Initial Configuration to execute $\Gamma \vdash N : C$

Γ	N	C	$\texttt{nil}$	C

Transitions

Γ	$\texttt{pm}\ M\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\}$	B	K	C	$\rightsquigarrow$
Γ	M	$A + A'$	$\texttt{pm}\ [\cdot]\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\} :: K$	C	
Γ	$\texttt{inl}\ P$	$A + A'$	$\texttt{pm}\ [\cdot]\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\} :: K$	C	$\rightsquigarrow$
Γ	$N[P/\texttt{x}]$	B	K	C	
Γ	πM	B	K	C	$\rightsquigarrow$
Γ	M	$B \times B'$	$\pi[\cdot] :: K$	C	
Γ	(N, N')	$B \times B'$	$\pi[\cdot] :: K$	C	$\rightsquigarrow$
Γ	N	B	K	C	
Γ	MN	B	K	C	$\rightsquigarrow$
Γ	M	$A \to B$	$[\cdot]N :: K$	C	
Γ	$\lambda\texttt{x}.P$	$A \to B$	$[\cdot]N :: K$	C	$\rightsquigarrow$
Γ	$P[N/\texttt{x}]$	B	K	C	
Γ	$\texttt{unfold}\ M$	$B[\mu X.B/X]$	K	C	$\rightsquigarrow$
Γ	M	$\mu X.B$	$\texttt{unfold}\ [\cdot] :: K$	C	
Γ	$\texttt{fold}\ N$	$\mu X.B$	$\texttt{unfold}\ [\cdot] :: K$	C	$\rightsquigarrow$
Γ	N	$B[\mu X.B/X]$	K	C	
Γ	$\texttt{choose}^{h}_{p \in P_h}\ M_p$	B	K	C	$\rightsquigarrow$
Γ	$M_{\hat{p}}$	B	K	C	$(\hat{p} \in P_h)$
Γ	$\texttt{input}^{o}_{i \in I_o}\ M_i$	B	K	C	$: \hat{\imath} =$
Γ	$M_{\hat{\imath}}$	B	K	C	$(\hat{\imath} \in I)$

Terminal Configurations

Γ	$\lambda\texttt{x}.P$	$A \to B$	$\texttt{nil}$	$A \to B$
Γ	(N, N')	$B \times B'$	$\texttt{nil}$	$B \times B'$
Γ	$\texttt{inl}\ M$	$A + A'$	$\texttt{nil}$	$A + A'$
Γ	$\texttt{fold}\ M$	$\mu X.B$	$\texttt{nil}$	$\mu X.B$
Γ	$\texttt{x}$	B	K	C

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

$$\frac{}{\Gamma|C \vdash^{\mathsf{k}} \mathtt{nil} : C} \qquad \frac{\Gamma|B \vdash^{\mathsf{k}} K : C}{\Gamma|B \times B' \vdash^{\mathsf{k}} \pi[\cdot] :: K : C}$$

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

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

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

sists of

- a finite or infinite sequence $s_0, s_1, \ldots$ in $R + O + \sum_{o \in O} I_o$—we call each $m < |s|$ a *move*, and more specifically, depending on s_m, 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 o-output move
- a function justifier mapping each arena move m to an earlier arena move or $*$, called the *justification pointer* from m, such that s_m is a child of $s_{\mathsf{justifier}(m)}$ (or a root if $\mathsf{justifier}(m) = *$).

We say that a move m is *described as* $n \curvearrowleft r$ where $n = \mathsf{justifier}(m)$ and $r = s_m$.

(ii) A *play* is a justified sequence s such that
- if m is an arena move, then justifier (m) is even if m is odd, and odd or $*$ if m is even
- if m is an output move, then m is even.

(iii) In a play s, a move m is a *Player-move* or an *Opponent-move* according as m is even or odd.

(iv) 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.

(v) An *nondeterministic infinite trace* (NIT) *strategy* σ 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 σ is a strategy on R. We write strat for the set of strategies on R.

(vi) 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. 2.4.

□

The following cartesian category of arenas is often used "behind the scenes" in game semantics, mainly for coherence isomorphisms. Here, we make it explicit.

Definition 4.2 (i) A *token-change* from arena R to arena S is a function $S \xrightarrow{f} R$, 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 elements of S strictly below b) to $R{\restriction_{fb}}$.

(ii) Given a token-change $R \xrightarrow{f} S$, and a strategy σ on S, we define a strategy $f..\sigma$ on R. Its finite traces, divergences and infinite traces are obtained by applying f to each element played in those of σ.

(iii) We write $\mathbf{TokCh}$ for the category of arenas and token-changes. Finite products are given by disjoint union, which we write as $\uplus$.

□

4.2 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 [1]. We also define a *left $\mathcal{G}$-module* i.e. functor from $\mathcal{G}^{\mathsf{op}}$ to $\mathbf{Set}$, that takes R to $\mathsf{strat}\ R$.

Definition 4.3 (i) For arenas R and S, we define $\mathcal{G}(R,S)$ to be $\prod_{b \in \mathsf{rt}\ S} \mathsf{strat}\ (R \uplus S{\restriction_b})$. We are writing $\mathsf{rt}\ S$ for the set of roots of S.

(ii) 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 $\mathsf{id}_{R,b}$.

□

In order to define composition, we need to define, for any arenas R, S, T, a map

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

Intuitively, the strategy $\sigma\backslash\tau$ should follow τ until that plays a root b of S, then continue in σ_b, until that plays another move in S, then follow τ again, and so forth. But the moves in S are hidden—"parallel composition with hiding".

Definition 4.4 (i) 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 $\mathsf{threads}\ s$. (f is a collection of *thread-pointers*.)

(ii) Let s be an interaction pre-sequence on R, S, T. For each $q \in \mathsf{threads}\ s$, we define the *arena* of s to be

Given an interaction pre-sequence s on R, S, T,

- its *outer thread*, a justified sequence on $R \uplus T$, consists of all moves in R and T, and all output and input moves
- its $*$*-inner thread*, a justified sequence on $S \uplus T$, 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 plays $b \in \mathsf{rt}\ S$, is a justified sequence on $R \uplus S_b$: it 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 q and subsequent input moves.

s is an *interaction sequence* when all these justified sequences are plays.

(iii) 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 4.5 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. Thus an interaction sequence may be

outer-Opponent-awaiting finite, with outer thread awaiting Opponent, and each inner thread awaiting Opponent

l-inner-Player awaiting finite, with outer thread and l-inner thread awaiting Player, and all other inner threads awaiting Opponent

outer-starved infinite, with outer thread awaiting Player, and each inner thread awaiting Opponent or infinite

outer-infinite infinite, with outer thread infinite, and each inner thread awaiting Opponent or infinite.

□

Definition 4.6 (i) Let R, S, T be arenas, let $\sigma \in \mathcal{G}(R, S)$ and let $\tau \in \mathsf{strat}\ (S \uplus T)$. For an interaction sequence s, and $q \in \mathsf{threads}\ s$, the "strategy for q" is τ or σ_b according as q is $*$ or plays $b \in \mathsf{rt}\ S$.

finite traces the outer thread of every outer-Opponent awaiting play s whose inner theads are finite traces of their strategies

divergences (1) the outer thread of every l-inner-Player awaiting play s whose l-inner thread is a divergence of its strategy, and whose other inner threads

are finite traces of their strategies

divergences (2) the outer thread of every outer-starved play whose inner threads are finite traces or infinite traces of their strategies

infinite traces the outer thread of every outer-infinite play whose inner threads are all finite traces or infinite traces of their strategies.

(ii) 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 $\sigma \backslash \tau_b$.

(iii) 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 4.7 Def. 4.6(ii) satisfies associativity and identity laws, making $\mathcal{G}$ a category. Def. 4.6(iii) satisfies associativity and left-identity laws, making strat a left $\mathcal{G}$-module. □

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.

The operation $\backslash$ can be recovered from the categorical structure:

Proposition 4.8 If $R \xrightarrow{\sigma} S$ and $R \uplus T \xrightarrow{\tau}$, then $\sigma \backslash \tau = (\sigma \times T); \tau$ □

By analogy with Prop. 2.5, we have

Proposition 4.9 • The operations $\backslash$ and token-changing and input^o all preserve determinism, and $\mathsf{id}_{R,b}$ is deterministic.

• Given signatures Z and Z', the hiding of

$$\begin{aligned}
\mathsf{id}_{R,b} &\text{ is } \mathsf{id}_{R,b} \\
f..\sigma &\text{ is } f..(\sigma \restriction Z) \\
\sigma \backslash \tau &\text{ is } (\sigma \restriction Z) \backslash (\tau \restriction Z) \\
\bigcup_{i \in I} \sigma_i &\text{ is } \bigcup_{i \in I} (\sigma_i \restriction Z)
\end{aligned}$$

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

where σ and τ and all σ_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 4.10 If $b \in \mathsf{rt}\ S$ and $R \xrightarrow{\sigma} S\restriction_b$, write $R \uplus S \xrightarrow{b \ltimes \sigma}$ for $\sigma \backslash (f..\mathsf{id}_{S,b})$, writing $S\restriction_b \uplus S \xrightarrow{f} S \uplus S\restriction_b$ for the obvious token-change. □

4.3 Model of Call-By-Name FPC

Definition 4.11 (i) A Q/A-*labelled arena* is an arena R, with every element classified as *question* or *answer*, where no answer enables an answer. It is Q-*rooted* when, moreover, every root is a question.

(ii) 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 ith root $\mathsf{root}\ i$. Similarly $\mathsf{pt}^{\mathsf{A}}_{i\in I} R_i$, provided that each R_i is Q-rooted.

(iii) Let R and S be Q/A-labelled arenas. We say that $R \sqsubseteq S$ when for every $r \in R$, both r and all its ancestors are elements of S, with the same labelling and parent-child relationship.

(iv) 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 ρ,

- $A \times B$ 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 Γ denotes R, and the type B denotes $\{S_j\}_{j\in J}$, then a term or a configuration inhabiting $\Gamma \vdash B$ denotes an element of $\prod_{j\in J} \mathsf{strat}\ (R \uplus S_j)$
- If the context Γ 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 is as follows. Let Γ denote R, and write $\dashrightarrow$ for token changing.

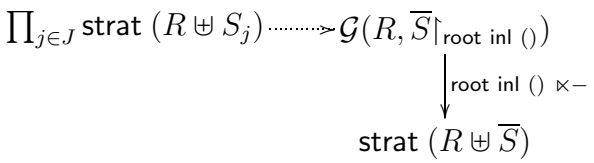
- $\mathtt{choose}^h$ and $\mathtt{input}^o$ are interpreted by $\bigcup$ and input^o.
- The operations of projection, pairing, λ, fold, unfold and stacking application, projection and unfold contexts are interpreted by token-changing.
- Suppose A denotes $\{S_j\}_{j\in J}$, and write $\hat{S}$ for $\mathsf{pt}^{\mathsf{Q}}_{j\in J} S_j$. Then $\Gamma, \mathtt{x} : A \vdash \mathtt{x} : A$ at j denotes

$$1 \xrightarrow{\mathsf{id}_{\hat{S},j}} \hat{S} \uplus \hat{S}\restriction_{\mathsf{root}\ j} \dashrightarrow (R \uplus \hat{S}) \uplus S_j$$

Other identifiers and $\mathtt{nil}$ are interpreted similarly.

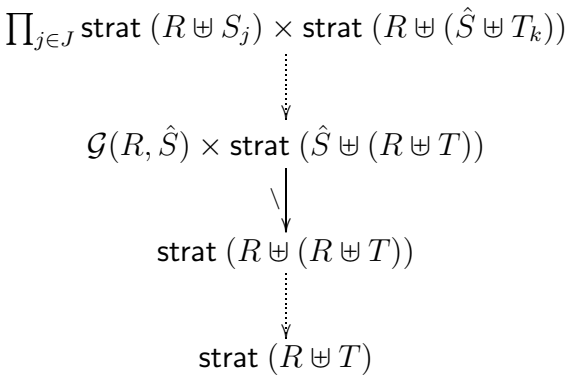
- Suppose A denotes $\{S_j\}_{j\in J}$ and A' denotes $\{S'_j\}_{j\in J'}$. Write $\overline{S}$ for $\mathsf{pt}^{\mathsf{A}}\{\mathsf{pt}^{\mathsf{Q}}_{i\in I} R_i, \mathsf{pt}^{\mathsf{Q}}_{j\in J} S_j\}$.

If $\Gamma \vdash M : A$, then $\mathsf{inl}\ M$ at $()$ denotes



applied to $[\![M]\!]$. And `inr` is interpreted similarly.

- Suppose A denotes $\{S_j\}_{j\in J}$ and B denotes $\{T_k\}_{k\in K}$, and write $\hat{S}$ for $\mathsf{pt}^{\mathsf{Q}}_{j\in J} S_j$. If $\Gamma \vdash M : A$ and $\Gamma \vdash N : A \to B$, then NM at k denotes



applied to $[\![M]\!], [\![N]\!]k$. The operations of pattern-match, stacking a pattern-match context and forming a configuration are interpreted similarly.

Definition 4.12 Let Y be a deadlock-free erratic signature, and Z an I/O signature. For a configuration d in $\mathcal{L}^{\mathsf{FPC}}(Y, Z, \Gamma \vdash C)$, where Γ 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 $[d]$ has a terminating trace sT, and t is a finite trace/divergence/infinite trace of $[\![T]\!]j$.

□

Using Prop. 4.9(4.9) and Prop. 4.8, we prove

Proposition 4.13 (i) If the erratic signature Y is empty, then the denotation of every term, stack and configuration d is deterministic, and so is $[\![d]\!]$.

(ii) [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

□

4.4 Computational Adequacy

Our task is to prove

Proposition 4.14 (adequacy) $[\![d]\!] = [\![d]\!]$, for every configuration d in $\mathcal{L}^{\mathsf{FPC}}(Y, Z, \Gamma \vdash C)$, where Y is deadlock-free. □

The basic plan is this. We define an *unhiding transform* that does two things

- add a ✓ to every step of execution
- turn each erratic choice into requested input

Thus, the transform of a configuration d, written $\overline{d}$, is deterministic and cannot diverge. It is easy to prove adequacy for such a term. Now if we take the denotation of $\overline{d}$, and hide both the ✓s and the requested inputs corresponding to erratic choice, we get back the denotation of d—that is because hiding commutes with composition. And the same goes for $[\![-]\!]$. So we deduce adequacy for d from that of $\overline{d}$.

We begin by stating as much of Prop. 4.14 as we are able in light of Prop. 4.13(ii):

Lemma 4.15 Let d inhabit $\Gamma \vdash C$, where $[\![C]\!] = \{S_j\}_{j \in J}$. Suppose $j \in J$.

(i) For a terminating trace $s(T, \mathtt{nil}\ )$ of $[d]$, and finite trace (divergence, infinite trace) t of $[\![T]\!]j$, the play st is a finite trace (divergence, infinite trace) of $[\![d]\!]j$.

(ii) Every finite trace of $[\![d]\!]j$ is a finite trace of $[\![d]\!]j$.

(iii) 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, stacks and configurations is defined in Fig. 3. The placing of ✓s is motivated by the decomposition in [11]—thunked subterms do not acquire a ✓.

Lemma 4.16 (i) If $\Gamma, \mathtt{x} : A \vdash M : B$ and $\Gamma \vdash N : A$ then $\overline{M[N/\mathtt{x}]} = \overline{M}[\overline{N}/\mathtt{x}]$.

(ii) Let $d = \Gamma, M, B, K, C$. If M is not `choose` or `input`, then either d and d' are both terminal, or $d \rightsquigarrow d'$ for unique d', and $\overline{d} \rightsquigarrow \checkmark.\overline{d'}$.

(iii) $[\overline{d}]$ has no divergences.

(iv) If $[d] = (A, B, C, D)$ then $[\overline{d}] \restriction Z = (A, B, C, \{s\overline{T} | sT \in D\})$

□

Now $[\![\overline{d}]\!]$ and $[\![\overline{d}]\!]$ are deterministic (Prop. 4.13(i)) and have the same finite traces (Lemma 4.15(ii)–(iii) and Lemma 4.16(iii)). So $[\![\overline{d}]\!] = [\![\overline{d}]\!]$. Prop. 4.9(4.9) tells us the following.

Lemma 4.17 (i) If P is a term or stack, then $[\![\overline{P}]\!] \restriction Z = [\![P]\!]$.

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

Fig. 3. The Unhiding Transform

(ii) If d is a configuration, $[\![\overline{d}]\!] \upharpoonright Z = [\![d]\!]$, so (by Lemma 4.16(iv)) $[\![\![\overline{d}]\!]\!] \upharpoonright Z = [\![\![d]\!]\!]$. □

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

5 Further Work

The adequacy proof above should be adapted to general references [1], and definability and full abstraction results formulated. It remains to characterize

(i) strategies definable with only countable choice

(ii) strategies definable without storage.

References

[1] Abramsky, S., 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.

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

[3] Danos, V., H. Herbelin and L. Regnier, *Game semantics and abstract machines*, in: *Proc., 11th Annual IEEE Symposium On Logic In Computer Science 1996*, 1996.

[4] Escardó, M., *A metric model of PCF* (1998), unpublished research note.

[5] Felleisen, M. and D. Friedman, *Control operators, the SECD-machine, and the λ-calculus*, in: M. Wirsing, editor, *Formal Description of Prog. Concepts*, North-Holland, 1986 .

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

[7] Hasegawa, M., "Models of sharing graphs :–a categorical semantics of let and letrec," Ph.D. thesis, University of Edinburgh (1997).

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

[9] Jonsson, B., *A fully abstract trace model for dataflow and asynchronous networks*, Distributed Computing **7** (1994), pp. 197–212.

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

[11] Levy, P. B., "Call-By-Push-Value. A Functional/Imperative Synthesis," Semantic Structures in Computation, Springer, 2004.

[12] Levy, P. B., *Adjunction models for call-by-push-value with stacks*, Theory and Applications of Categories **14** (2005), pp. 75–110.

[13] McCusker, G., "Games and Full Abstraction for a Functional Metalanguage with Recursive Types," Ph.D. thesis, University of London (1996).

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

[15] Plotkin, G., *Domains* (1983), prepared by Y. Kashiwagi, H. Kondoh and T. Hagino.

[16] Plotkin, G. and J. Power, *Notions of computation determine monads*, in: *Proc., Foundations of Software Sci. and Comp. Struct., 2002*, LNCS **2303** (2002).

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

[18] Roscoe, A. W., *Seeing beyond divergence* (2004), presented at BCS FACS meeting "25 Years of CSP".