Infinite Trace Equivalence

Paul Blain Levy

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, in a call-by-push-value setting, by adapting earlier game models. Traditional adequacy proofs using logical relations are not applicable, so we use instead a novel hiding argument. Finally, we give junk-freeness and full abstraction results that hold under limited circumstances.

Key words: nondeterminism, infinite traces, game semantics, call-by-push-value

1 Introduction

1.1 The Problem

Consider the following call-by-name [1] 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.

Email address: pbl@cs.bham.ac.uk (Paul Blain Levy).
URL: www.cs.bham.ac.uk/~pbl (Paul Blain Levy).

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

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. To illustrate this very natural notion of equivalence, consider the following properties that appear in the specification for a program called PROG.

safety PROG must not kill the customer.
liveness PROG must (eventually) greet the customer.
conditional liveness If PROG insults the customer, it must (eventually) apologize.
infinite liveness PROG must (eventually) stop insulting the customer.

If we know PROG's infinite trace equivalence class—i.e. its range of behaviours—then we know which of these conditions are satisfied.

As stated in (Plotkin, 1983), "we [. . .] desire a semantics such that [a term's denotation] 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 (Plotkin, 1983) and divergence semantics (Roscoe, 2004), identify programs that are not infinite trace equivalent, so they are too coarse. In particular, they cannot identify whether a program satisfies all four of the above conditions. Other styles of semantics count the internal manipulations (Brookes, 2002; Escardó, 1998) or include branching-time information (Cattani and Winskel, 1997; Panangaden and Russell, 1989), 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 (Hyland and Ong, 2000), 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 (McCusker, 1996).

1.2 Why Explicit Forcing?

Before turning to our solution, we consider two kinds of semantics that have been studied. We abbreviate $\mathtt{print}\ c$ to c.

(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 $\bot$. Ex-

amples are the Hoare, Smyth and Plotkin powerdomain semantics (Plotkin, 1983), all the CSP semantics in (Roscoe, 1998), and the game semantics of (Harmer and McCusker, 1999). Divergence-least semantics cannot model infinite trace equivalence, by the following argument taken from (Plotkin, 1983).

$$\begin{aligned} \text{Put}\quad & M = \bot \texttt{ or } \mathsf{ins}.\mathsf{apol}.\bot \\ & M' = \bot \texttt{ or } \mathsf{ins}.\bot \texttt{ or } \mathsf{ins}.\mathsf{apol}.\bot \\ \text{Then}\quad & M = \bot \texttt{ or } \bot \texttt{ or } \mathsf{ins}.\mathsf{apol}.\bot \leqslant M' \\ & M = \bot \texttt{ or } \mathsf{ins}.\mathsf{apol}.\bot \texttt{ or } \mathsf{ins}.\mathsf{apol}.\bot \geqslant M' \end{aligned}$$

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

In particular, if M insults the customer, then it must apologize, but this is not true of M'. Therefore, divergence-least semantics cannot verify conditional liveness properties—by contrast with divergence semantics (Roscoe, 2004).

(2) A *well-pointed* semantics is one where (roughly speaking) a term denotes a function from the set of *environments*. Examples are the 3 powerdomain semantics (Plotkin, 1983), all the CSP semantics in (Roscoe, 1998), the semantics using infinite traces in (Brookes, 2002), and divergence semantics (Roscoe, 2004). 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. Suppose that $\mathcal{A}$ contains just one character ins, and consider the following two terms [2] involving $\mathtt{x}$.

$$\begin{aligned} N &= (\texttt{choose } n.\ \mathsf{ins}^n.\ \mu\mathtt{z}.\mathtt{z}) \texttt{ or } \mathtt{x} \\ N' &= (\texttt{choose } n.\ \mathsf{ins}^n.\ \mu\mathtt{z}.\mathtt{z}) \texttt{ or } \mathsf{ins}.\ \mathtt{x} \end{aligned}$$

On the one hand, N and N' are infinite trace equivalent in every environment, because ins.

closed term	$N[M/\mathtt{x}]$	$N'[M/\mathtt{x}]$
can print ins^n then diverge	yes	yes
can print ins^ω	iff M can print ins^ω	iff M can print ins^ω

On the other hand, they are not contextually equivalent:

closed term	$\mu\mathtt{x}.N$	$\mu\mathtt{x}.N'$
can print ins^n then diverge	yes	yes
can print ins^ω	no	yes

[2] discovered by A. W. Roscoe in 1989 [personal communication], and independently in (Levy, 2004b).

and so any model of infinite trace equivalence must distinguish them. In particular, $\mu\mathtt{x}.N$ must stop insulting the customer, but that is not the case for $\mu\mathtt{x}.N'$. Therefore, divergence semantics (Roscoe, 2004), which identifes N and N', cannot verify infinite liveness.

(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 $\mathtt{choose}^{\perp}\ n_{\in\mathbb{N}}.M_n$ to be $(\mu\mathtt{f}\lambda n.(M_n\ \mathtt{or}\ \mathtt{f}(n+1)))0$, which either executes some M_n or diverges (Plotkin, 1983). Using this instead of $\mathtt{choose}\ n_{\in\mathbb{N}}$, the same problem arises.)

A naive way of distinguishing N and N' is to say that N' is able to print a tick and then force (i.e. execute) $\mathtt{x}$, whereas N is not:

term involving $\mathtt{x}$	N	N'
can print ins^n then diverge	yes	yes
can print ins^ω	no	no
can force $\mathtt{x}$	yes	no
can print ins then force $\mathtt{x}$	no	yes
can print ins^{n+2} then force $\mathtt{x}$	no	no

And that gives our solution.

This idea, that a model of call-by-name should make explicit when a program forces its (thunked) argument, is present—often implicitly—in game semantics, where (as argued in (Levy, 2004a)) "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 (Harmer and McCusker, 1999), 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.

Consider, for example, the two (call-by-name) terms

$$\begin{aligned} P &= \lambda\mathtt{x}.(\mathtt{diverge}\ \mathtt{or}\ \mathtt{if}\ \mathtt{x}\ \mathtt{then}\ (\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{true})\ \mathtt{else}\ \mathtt{true}) \\ P' &= \lambda\mathtt{x}.(\mathtt{diverge}\ \mathtt{or}\ (\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{diverge}\ \mathtt{else}\ \mathtt{true}) \\ &\qquad \mathtt{or}\ \mathtt{if}\ \mathtt{x}\ \mathtt{then}\ (\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{true})\ \mathtt{else}\ \mathtt{true}) \end{aligned}$$

of type $\mathtt{bool} \to \mathtt{bool}$. In (Harmer and McCusker, 1999), these terms have the same denotation, and indeed are observationally equivalent for may and must testing. But if we add printing to the language, then we can place these terms in the ground context

$$\mathcal{C}[\cdot] = [\cdot](\checkmark.\mathtt{true})$$

Now $\mathcal{C}[P]$ may print a tick and then diverge, whereas $\mathcal{C}[P']$ may not. Therefore, from the viewpoint of infinite trace equivalence, P and P' must have different denotations. We shall see that this is the case in our model.

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 to provide 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. Then we describe the model, and prove it adequate. Whereas in (Levy, 2006) we model a call-by-name calculus (in keeping with most of the game literature), in this paper we model *call-by-push-value*, a calculus that contains both call-by-name and call-by-value as fragments. This has some advantages:

- it makes explicit the two kinds of moves in game semantics: forcing ("asking a question") and returning ("answering")
- its simple categorical structure based on an adjunction (Levy, 2005) contrasts with the rather awkward structure of call-by-name
- the *unhiding transform* used in the adequacy proof is more straightforward than for call-by-name and call-by-value.

Finally, we obtain some limited definability and full abstraction properties of the model—by extending the language, rather than by cutting down the model.

1.4 Related Work: Dataflow Networks

An infinite trace model for *dataflow networks*—including feedback, but not recursion—was presented in (Jonsson, 1994), and shown fully abstract. In the terminology of (Hasegawa, 1997), it forms a *cartesian-centre traced symmetric monoidal category*.

Although it is shown in (Hasegawa, 1997) that such a category, if *centrally closed*, can be converted into a kind of recursion, that is not useful here because Jonsson's model is not centrally closed. (Nor, for that matter, is its finite trace variant.)

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

In this section, we extend the language of Sect. 1.1 to allow choice operators of arbitrary arity. But *empty* arity presents a problem: a language containing a command "choose an element of the empty set" cannot be implemented. To skirt this problem, we introduce the notion of an *omni-error*: if u is an omni-error for a programming language, then any program at any time can abort by printing `Omni-error message` u.

Definition 1 An *erratic signature* is a family of sets $\{P_h\}_{h\in H}$. It is *deadlock-free* with respect to a set U of omni-errors when either all P_h are nonempty or U is nonempty. □

Any such $\{P_h\}_{h\in H}$—together with an alphabet $\mathcal{A}$—determines a language $\mathcal{L}(Y,\mathcal{A})$ in which for each $h \in H$, there is a construct $\mathtt{choose}^h\{M_p\}_{p\in P_h}$ that erratically chooses $p \in P_h$ and then executes M_p. The syntax is

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

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

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

Given a set U of omni-errors, we also write $M \overset{u}{\rightsquigarrow}$ for every closed term M and $u \in U$.

Usually, U would be empty, in which case 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 the deadlock-free 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 $[M]_U \subset (\mathcal{A}^* + \mathcal{A}^\omega) + \mathcal{A}^* \times U$ 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 (1)–(2), 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 1 (1) For some deadlock-free signatures and alphabets, infinite trace equivalence is strictly finer than the kernel of $[-]_{\mathsf{inf}}$.
(2) For all deadlock-free 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 deadlock-freeness, every finite path extends to a path that is either infinite or ends in an omni-error. □

Prop. 1(2) legitimates leaving omni-errors out of a semantics of infinite trace equivalence (in the deadlock-free situation), 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 construct $\mathtt{input}^o\{M_i\}_{i \in I_o}$ that prints o, then waits for the user to supply some $i \in I_o$, and then executes M_i. The *cardinality* of Z, written $|Z|$, is $|O + \sum_{o \in O} I_o|$.

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^{det} 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 (Moggi,

1991; Plotkin and Power, 2002), to model requested input. Note that this includes as special cases the monads designated in (Moggi, 1991) as "interactive input", "interactive output", and "exceptions". We accordingly say that $o \in O$ is a *printing operator* when I_o is singleton, and an *error operator* when I_o is empty. This allows us to dispense with $\mathcal{A}$, regarding each $c \in \mathcal{A}$ as a printing operator in Z, and regarding $\texttt{print}\ c.\ M$ as syntactic sugar[3] for $\texttt{input}^c\{M\}_{i\in 1}$.

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. For what should the actions be? On the one hand, if we allow both outputs and inputs to be actions, we need additional alternation and receptivity-to-input conditions. On the other hand, 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 (BLTS) Let $Z = \{I_o\}_{o\in O}$ be an I/O signature.

(1) A *bi-labelled transition system* (BLTS) $\mathcal{L}$ over Z and U consists of
- 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* wrt a set U of omni-errors when either every silent state has at least one successor or U is nonempty.

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

(3) A terminable BLTS is *deterministic* when each silent state has precisely one successor.

□

As with LTS's, we can obtain trace sets of states. To begin with, we need to characterize these trace sets as "strategies".

[3] The reader may feel that there is a substantial difference between these two things, because $\texttt{print}\ c.\ M$ prints c and immediately executes M, whereas $\texttt{input}^c\{M\}_{i\in 1}$ prints c and then waits for a response before continuing to execute M, and if no response is received it never executes M. However, it would appear that this difference is denotationally immaterial, at least in the sequential setting.

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 over Z terminating in V* is a Player-awaiting play extended with an element of V.

(2) A *nondeterministic infinite trace* (NIT) *strategy over Z into V* is a tuple $\sigma = (A, B, C, D)$ where
finite traces A is a set of input-awaiting plays
divergences B is a set of Player-awaiting plays
infinite traces C is a set of infinite plays
terminating traces D is a set of plays terminating in V
such that if t is in A, B, C or D, then every input-awaiting prefix of t is in A.

(3) A Player-awaiting play t is *finitely consistent* with a NIT strategy σ when every input-awaiting prefix of t is a finite trace of σ.

(4) A NIT strategy $\sigma = (A, B, C, D)$ is *deterministic* when
- any Player-awaiting play t finitely consistent with σ 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 t whose input-awaiting prefixes are in σ is in C.

(5) Let s be a Player-awaiting play over Z into V. A NIT strategy over Z into V *starting from* s is a set $\sigma = (A, B, C, D)$ of plays extending s such that all finite prefixes of these plays that extend s are in A. We define *determinism* for such strategies as in (4).

(6) (adapted from (Roscoe, 1998)) A NIT strategy σ into V is *deadlock-free* wrt a set U when either U is nonempty or, for every Player-awaiting s' that is finitely consistent with σ, there is a deterministic strategy τ starting from s' such that $\tau \subseteq \sigma$.

(7) We write $T_{UZ}\, V$ for the set of all NIT strategies over Z into V deadlock-free wrt U. (Thus, if U is nonempty, it will consist of *all* the NIT strategies over Z wrt U.) Clearly T_{UZ} is a strong monad on $\mathbf{Set}$.

□

Here are some operations on strategies.

Definition 4 (1) For $d \in V$, we define ηd (the monad's unit at d) to be the deterministic strategy

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

(2) 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

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

(3) 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

$\mathsf{input}^o\{\sigma_i\}_{i\in I_o}$ for the strategy

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

□

Proposition 2 (1) A deterministic strategy is deadlock-free wrt any set U.
(2) input^o preserves determinism, and preserves deadlock-freedom wrt any set U.
(3) If I or U is nonempty, then $\bigcup_{i\in I}$ preserves deadlock-freedom wrt U.

□

Definition 5 (BLTS to strategies) Let Z be an I/O signature, and let $\mathcal{L}$ be a terminable BLTS over 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 strategy $(A, B, C, D) \in T_{UZ}\ V$ 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 Let d be a state in a terminable BLTS $\mathcal{L}$ over Z.

(1) If $\mathcal{L}$ is deterministic, then so is $[d]$.
(2) If $\mathcal{L}$ is deadlock-free wrt U, then so is $[d]$.

□

Proposition 4 Let d be a state in a terminable BLTS over Z.

- If d is an o-state then $[d] = \mathsf{input}^o\{[d : i]\}_{i\in I_o}$
- If d is a silent state then $[d] = \bigcup_{d\rightsquigarrow d'}[d']$
- If d is a terminal state then $[d] = \eta d$.

□

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

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

For each context $\Gamma = \mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$, we write $\mathcal{L}(Y, Z, \Gamma)$ for the set of terms $\Gamma \vdash M$. This set is a terminable BLTS over Z, deadlock-free wrt U, in which

- every identifier is terminal
- $\mu\mathtt{x}.M$ is silent, and $\mu\mathtt{x}.M \rightsquigarrow M[\mu\mathtt{x}.M/\mathtt{x}]$
- $\mathtt{choose}^h\{M_p\}_{p\in P_h}$ is silent, and $\mathtt{choose}^h\{M_p\}_{p\in P_h} \rightsquigarrow M_{\hat{p}}$ for each $\hat{p} \in P_h$
- $\mathtt{input}^o\{M_i\}_{i\in I_o}$ is an o-state, and $(\mathtt{input}^o\{M_i\}_{i\in I_o}) : \hat{\imath} = M_{\hat{\imath}}$ for each $\hat{\imath} \in I_o$

The following is trivial.

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 $M[N/\mathtt{x}] = \mathtt{y}$.

□

From this we can deduce the key result: we can characterize $[-]$ in a compositional way:

Proposition 5 In the language $\mathcal{L}(Y, Z)$, we have

(1) If $\mathtt{x} \in \Gamma$, then $[\Gamma \vdash \mathtt{x}] = \eta\mathtt{x}$
(2) $[\Gamma \vdash \mathtt{choose}^h\{M_p\}_{p\in P_h}] = \bigcup_{p\in P_h}[\Gamma \vdash M_p]$
(3) $[\Gamma \vdash \mathtt{input}^o\{\Gamma M_i\}_{i\in I_o}] = \mathsf{input}^o\{[\Gamma \vdash M_o]\}_{i\in I_o}$
(4) If $\Gamma, \mathtt{x} \vdash M$ then

$$[\Gamma \vdash \mu\mathtt{x}.M] = \mu[\Gamma, \mathtt{x} \vdash M]$$

where we define $\mu(A, B, C, D)$ to be

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

(5) If $\Gamma, \mathtt{x} \vdash M$ and $\Gamma \vdash N$, then

$$[\Gamma \vdash M[N/\mathtt{x}]] = [\Gamma, \mathtt{x} \vdash M] * [\Gamma \vdash N]$$

where we define $(A, B, C, D) * (A', B', C', D')$ to be

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

□

Prop. 5 gives us a computationally adequate denotational semantics.

Proposition 6 The operations μ and $*$ defined in Prop. 5 preserve determinism, and preserve deadlock-freedom wrt any set U. □

2.5 Hiding

We first define the notion of hiding on BLTS's, then adapt it to strategies. Let $Z = \{I_o\}_{o \in O}$ and $Z' = \{I'_o\}_{o \in O'}$ be I/O signatures.

Definition 6 Let $\mathcal{L}$ be a terminable BLTS over $Z + Z'$. The *hiding* of Z' in $\mathcal{L}$, written $\mathcal{L} \setminus Z'$, is the BLTS wrt Z obtained from $\mathcal{L}$ by making each o-state (where $o \in Z'$) silent, with a transition to $d : i$ for every $i \in I_o$. □

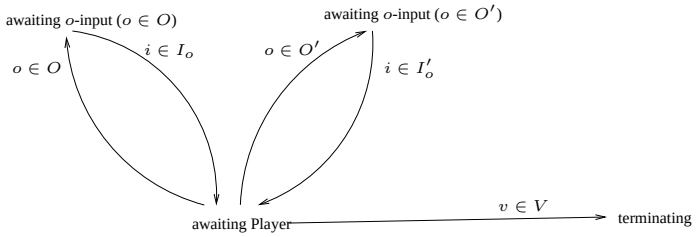
For strategies, we must begin with plays:

Proposition 7 Let s be a play wrt $Z + Z'$ into V. Write $s \setminus Z'$ for the play wrt Z into V obtained by suppressing all I/O moves in Z'. Then precisely one of the following hold:

awaiting outer input s and $s \setminus Z'$ await o-input, where $o \in Z$
awaiting inner Player s and $s \setminus Z'$ await Player
awaiting hidden input s awaits o-input, where $o \in Z'$, and $s \setminus Z'$ awaits Player
outer starved s is infinite and $s \setminus Z'$ awaits Player.
outer infinite s and $s \setminus Z'$ are infinite
outer terminating s and $s \setminus Z'$ are terminating

(Here, "outer" refers to $s \setminus Z'$ and "inner" refers to s.) □

This is proved by induction for finite plays, which obey the state diagram



and it is then trivial for the infinite plays.

Definition 7 Given a strategy $\sigma = (A, B, C, D)$ into V wrt $Z + Z'$, the *hiding* of σ, written $\sigma \setminus Z'$, is the strategy wrt Z defined as follows

finite traces $s \setminus Z'$, where s awaits outer input and is a finite trace of σ
divergences (1) $s \setminus Z'$, where s awaits inner Player and is a divergence of σ
divergences (2) $s \setminus Z'$, where s is outer starved and is an infinite trace of σ
infinite traces $s \setminus Z'$, where s is outer infinite and is an infinite trace of σ
terminating traces $s \setminus Z'$, where s is outer terminating and is a terminating trace of σ.

□

Proposition 8 If d is a state in a BLTS $\mathcal{L}$ over $Z + Z'$ then

$$([d]_{\mathcal{L}}) \setminus Z' = [d]_{(\mathcal{L} \setminus Z')}$$

□

Proposition 9 Given signatures Z and Z', the hiding of

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

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

Anything BLTS or strategy can be obtained by the application of hiding to a deterministic one.

Proposition 10 Let Z be an I/O signature.

(1) For every terminable BLTS $\mathcal{L}$ over Z, there exists an I/O signature Z' and a deterministic terminable BLTS $\mathcal{L}'$ over $Z + Z'$ such that $\mathcal{L} = \mathcal{L}' \setminus Z'$.
(2) There exists an I/O signature Z' with the following property. For every strategy σ over Z into a countable set V, deadlock-free wrt a set U, there exists a deterministic strategy τ over $Z + Z'$ into V such that $\tau \setminus Z' = \sigma$.

□

Proof (1) Define Z' to provide an operator for each silent state d of $\mathcal{L}$, with arity $\{d' | d \rightsquigarrow d'\}$. Then construct $\mathcal{L}'$ from $\mathcal{L}$ by making each silent state d into a d-state, with $d : d' = d'$.

(2) We define Z' to be the I/O signature with two operators o and o', where I_o is empty, and $I_{o'} = 2^{\mathsf{max}(\aleph_0, |Z|)}$. Given $\sigma = (A, B, C)$, there exists $U \subseteq I_{o'}$ and a surjection $U \xrightarrow{f} A + B + C$. For each $i \in I_{o'}$, define a deterministic strategy $g(i)$. If $i \notin U$, then $g(i)$ is $\mathsf{input}^o\{\}$. If $i \in U$, then $g(i)$ is the following deterministic strategy over $Z + Z'$:

$$(\{t | t \text{ awaits Opponent}, t \sqsubseteq s\} \cup \{tmo | t \text{ awaits Opponent}, t \sqsubseteq s, tm \not\sqsubseteq s\},$$
$$\{s | s \text{ awaits Player}\},$$
$$\{s | s \text{ is infinite }\})$$

Define τ to be the deterministic strategy

$$\mathsf{input}^{o'}\{g(i)\}_{i \in I(o')}$$

It is then clear that $\tau \setminus Z' = \sigma$. □ The operation used in (1) is called *unhiding*.

3 Call-By-Push-Value With Type Recursion

We now want to move to a language with higher order and recursive types, that contains $\mathcal{L}(Y, Z)$ as a fragment. Again let $Y = \{P_h\}_{h \in H}$ be an erratic signature deadlock-free wrt U, and let $Z = \{I_o\}_{o \in I}$ be an I/O signature. The calculus we use is *call-by-push-value* (CBPV) (Levy, 1999), which we now review.

CBPV has two disjoint classes of terms: values and computations. It likewise has two disjoint classes of types: a value has a value type, while a computation has a

computation type. For clarity, we underline computation types. The types (including recursive types) are given by

$$\text{value types} \qquad A ::= \quad U\underline{B} \mid \textstyle\sum_{i\in I} A_i \mid 1 \mid A \times A \mid X \mid \mu X.A$$

$$\text{computation types } \underline{B} ::= \quad FA \mid \textstyle\prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \mid \mu \underline{X}.\underline{B}$$

where I is any countable set. We omit rules for the type 1 as they are analogous to those for $\times$.

The easiest way to understand these types is via their Scott semantics: a value type denotes a cpo, and a computation type denotes a *pointed* cpo (one with a least element). In particular, FA denotes the lift of $[\![A]\!]$, and $U\underline{B}$ denotes the same cpo as $\underline{B}$. The $\sum$ connective denotes disjoint union of cpos, and $A \to \underline{B}$ denotes the pointed cpo of continuous functions from $[\![A]\!]$ to $[\![\underline{B}]\!]$. More generally, given a strong monad T on a cartesian category $\mathcal{C}$—and suitable coproducts, products and exponentials—a value type denotes a $\mathcal{C}$-object and a computation type denotes a T-algebra. (However, the game semantics we shall look at is not of this kind.)

The typing rules, including introduction/elimination rules for each connective, and rules for nondeterminism using Y and I/O using Z, are given in Fig. 3. Here `let` is used to bind an identifier, `pm` is used to pattern-match, and the ‘ symbol represents application written operand first. Note that $\lambda \mathtt{x}.M$ is a computation, not a value, and function type is a computation type; this distinguishes CBPV from call-by-value.

As for call-by-name and call-by-value, the operational semantics of CBPV can be presented in a number of styles: a big-step semantics appears in (Levy, 1999), but here *CK-machine* semantics (Felleisen and Friedman, 1986) is more suitable. The machine is shown as a terminable BLTS in Fig. 2–3. As we work on a term, we keep a stack of contexts, similar to an evaluation context. For example, to evaluate M `to x.` N, we first need to evaluate M, so the rest of the term—the context $[\cdot]$ `to x.` N—is placed onto the stack. Later, when M has been evaluated, this context is removed from the stack and used.

We write $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ to mean that K is a stack that can accompany a computation inhabiting $\Gamma \vdash^{\mathsf{c}} \underline{B}$ in the course of evaluating a computation inhabiting $\Gamma \vdash^{\mathsf{c}} \underline{C}$. The typing rules for this judgement are defined inductively in Fig. 4.

Definition 8 A *configuration* inhabiting $\Gamma \vdash^{\mathsf{c}} \underline{C}$ consists of

- a computation type $\underline{B}$
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$
- a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$.

□

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} \langle \hat{\imath}, V \rangle : \sum_{i \in I} A_i}\ \hat{\imath} \in I \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B}\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . M_i\}_{i \in I} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} \langle V, V' \rangle : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle . M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \lambda\{i.M_i\}_{i \in I} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} \hat{\imath}\text{`}M : \underline{B}_{\hat{\imath}}}\ \hat{\imath} \in I$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} V\text{`}M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A[\mu \mathtt{X}.A/\mathtt{X}]}{\Gamma \vdash^{\mathsf{v}} \mathtt{fold}\ V : \mu \mathtt{X}.A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mu \mathtt{X}\ A \quad \Gamma, \mathtt{x} : A[\mu \mathtt{X}.A/\mathtt{X}] \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}[\mu \underline{\mathtt{X}}.\underline{B}/\underline{\mathtt{X}}]}{\Gamma \vdash^{\mathsf{c}} \mathtt{fold}\ M : \mu \underline{\mathtt{X}}.\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \mu \underline{\mathtt{X}}.\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{unfold}\ M : \underline{B}[\mu \underline{\mathtt{X}}.\underline{B}/\underline{\mathtt{X}}]}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M_p : \underline{B}\ \ (\forall p \in P_h)}{\Gamma \vdash^{\mathsf{c}} \mathtt{choose}^h\{M_p\}_{p \in P_h} : \underline{B}}\ h \in H \qquad \frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}\ \ (\forall i \in I_o)}{\Gamma \vdash^{\mathsf{c}} \mathtt{input}^o\{M_i\}_{i \in I_o} : \underline{B}}\ o \in O$$

Fig. 1. Syntax of CBPV with Type Recursion, Erratic Choice and I/O

We write $\mathcal{L}^{\mathsf{CBPV}}(Y, Z, \Gamma \vdash \underline{C})$ for the terminable BLTS over Z, deadlock-free wrt U, whose states are the configurations inhabiting $\Gamma \vdash^{\mathsf{c}} \underline{C}$, and which is defined in Fig. 2–3.

Note Formally, all terms are deemed to be explicitly typed. However, to reduce

Initial configuration		for evaluation of $\Gamma \vdash^{\mathsf{c}} M : \underline{C}$			
Γ	M	$\underline{C}$	$\mathtt{nil}$	$\underline{C}$	
Silent Configurations And Their Transitions					
Γ	$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$M[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$	
Γ	$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	M	FA	$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N :: K$	$\underline{C}$	
Γ	$\mathtt{return}\ V$	FA	$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N :: K$	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$N[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$	
Γ	$\mathtt{force}\ \mathtt{thunk}\ M$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	M	$\underline{B}$	K	$\underline{C}$	
Γ	$\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{(i, \mathtt{x})\}_{i\in I}$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$M_{\hat{\imath}}[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$	
Γ	$\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$M[V/\mathtt{x}, V'/\mathtt{y}]$	$\underline{B}$	K	$\underline{C}$	
Γ	$\hat{\imath}\text{`}M$	$\underline{B_{\hat{\imath}}}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	M	$\prod_{i\in I}\underline{B_i}$	$\hat{\imath} :: K$	$\underline{C}$	
Γ	$\lambda\{i.M_i\}_{i\in I}$	$\prod_{i\in I}\underline{B_i}$	$\hat{\imath} :: K$	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$M_{\hat{\imath}}$	$\underline{B_{\hat{\imath}}}$	K	$\underline{C}$	
Γ	$V\text{`}M$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	M	$A \to \underline{B}$	$V :: K$	$\underline{C}$	
Γ	$\lambda\mathtt{x}.M$	$A \to \underline{B}$	$V :: K$	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$M[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$	

Fig. 2. CK-Machine For CBPV, With Types

clutter, we have not actually written the types.

We embed

$$\mathcal{L}(Y, Z, (\mathtt{x}_0, \ldots, \mathtt{x}_{n-1})) \quad \text{in} \quad \mathcal{L}^{\mathsf{CBPV}}(Y, Z, (\mathtt{x}_0 : UF0, \ldots, \mathtt{x}_{n-1} : UF0 \vdash F0))$$

in the standard manner of translating CBN into CBPV (Levy, 1999), i.e. $\mathtt{x}_i$ gets transformed into $\mathtt{force}\ \mathtt{x}_i$.

Silent Configurations And Their Transitions (continued)

Γ	$\texttt{unfold}\ M$	$\underline{B}[\mu\underline{X}\underline{B}/\underline{X}]$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	M	$\mu\underline{X}\underline{B}/\underline{X}$	$\texttt{unfold} :: K$	$\underline{C}$	
Γ	$\texttt{fold}\ N$	$\mu\underline{X}\underline{B}/\underline{X}$	$\texttt{unfold} :: K$	$\underline{C}$	silent, $\rightsquigarrow$
Γ	N	$\underline{B}[\mu\underline{X}\underline{B}/\underline{X}]$	K	$\underline{C}$	
Γ	$\texttt{pm fold}\ V\ \texttt{as fold x}.\ M$	$\underline{B}$	K	$\underline{C}$	silent, $\rightsquigarrow$
Γ	$M[V/\texttt{x}]$	$\underline{B}$	K	$\underline{C}$	
Γ	$\texttt{choose}^h\{M_p\}_{p\in P_h}$	B	K	C	silent, $\rightsquigarrow$
Γ	$M_{\hat{p}}$	$\underline{B}$	K	$\underline{C}$	$(\hat{p} \in P_h)$

Output Configurations And Their Transitions

Γ	$\texttt{input}^o\{M_i\}_{i\in I_o}$	$\underline{B}$	K	$\underline{C}$	o output, $:\hat{\imath} =$
Γ	$M_{\hat{\imath}}$	$\underline{B}$	K	$\underline{C}$	$(\hat{\imath} \in I)$

Terminal configurations

Γ	$\texttt{return}\ V$	FA	$\texttt{nil}$	FA	terminal
Γ	$\lambda\{i.M_i\}_{i\in I}$	$\prod_{i\in I}\underline{B}_i$	$\texttt{nil}$	$\prod_{i\in I}\underline{B}_i$	terminal
Γ	$\lambda\texttt{x}.M$	$A \to \underline{B}$	$\texttt{nil}$	$A \to \underline{B}$	terminal
Γ	$\texttt{fold}\ M$	$\mu\underline{X}.\underline{B}$	$\texttt{nil}$	$\mu\underline{X}.\underline{B}$	terminal
Γ	$\texttt{force z}$	$\underline{B}$	K	$\underline{C}$	terminal
Γ	$\texttt{pm z as}\ \{(i,\texttt{x}).M_i\}_{i\in I}$	$\underline{B}$	K	$\underline{C}$	terminal
Γ	$\texttt{pm z as}\ (\texttt{x},\texttt{y}).M$	$\underline{B}$	K	$\underline{C}$	terminal
Γ	$\texttt{pm z as fold x}.\ M$	$\underline{B}$	K	$\underline{C}$	terminal

Fig. 3. CK-Machine For CBPV, With Types (continued)

4 Pointer Games

4.1 Pointer Game On Arena

We obtain our model of CBPV by taking the standard game semantics of (McCusker, 1996)—omitting for simplicity, the constraints of innocence, visibility and bracketing, although the latter two could easily be incorporated—reformulated for CBPV and removing the nondeterminism constraint in the manner of Def. 3(2).

$$\frac{}{\Gamma|\underline{C} \vdash^{\mathsf{k}} \mathtt{nil} : \underline{C}} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|FA \vdash^{\mathsf{k}} [\cdot]\, \mathtt{to}\, \mathtt{x}.\, M :: K : \underline{C}}$$

$$\frac{\Gamma|\underline{B}_{\hat{\imath}} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|\prod_{i\in I}\underline{B}_i \vdash^{\mathsf{k}} \hat{\imath} :: K : \underline{C}}\ \hat{\imath} \in I \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|A \to \underline{B} \vdash^{\mathsf{k}} V :: K : \underline{C}}$$

$$\frac{\Gamma|\underline{B}[\mu\underline{X}.\underline{B}/\underline{X}] \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|\mu\underline{X}.\underline{B} \vdash^{\mathsf{k}} \mathtt{unfold} :: K : \underline{C}}$$

Fig. 4. Typing rules for stacks

We begin with arenas.

Definition 9 (1) Let R be set equipped with a relation $\vdash \subseteq (\{*\} + \mathsf{moveset}) \times \mathsf{moveset}$. The *roots* of R are $\mathsf{rt}\ R = \{r | * \vdash r\}$ $\mathsf{rt}\ R \subseteq R$, while the *children* of $s \in S$ are $\{r | s \vdash r\}$. We say that R is a *forest* when all these subsets are disjoint, and, for every $r \in R$ there is a (necessarily unique) finite sequence

$$* \vdash r_0 \vdash \cdots \vdash r_n = r$$

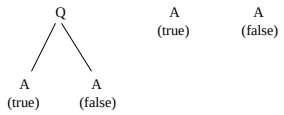
(2) An *arena* is a countable forest. (We do not require Q/A labelling on elements of R, at this stage, because we are not imposing the bracketing condition.)
(3) We write $R \uplus S$ for the disjoint union of R and S.
(4) If $r \in R$, we write $R\restriction_s$ for the arena of elements *strictly* descended from r.
(5) A *token renaming* from arena R to arena S is a function $R \xrightarrow{f} S$, such that, if $b \in \mathsf{rt}\ R$, then $fb \in \mathsf{rt}\ S$ and f restricts to an arena isomorphism $R\restriction_b \cong S\restriction_{fb}$. We write $\mathbf{TokRen}$ for the category of arenas and token renamings. This has finite coproducts given by disjoint union (and indeed countable coproducts, though we do not use these).

□

Given an arena R, the *pointer game* on R is informally described as follows.

- 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.

For example, a possible play for the pointer game on the arena



is

(1)

In fact, this play will distinguish the terms P and P' given in Sect.1.2. In the case of P', the denotation includes (1) as a divergence, whereas in the case of P it does not.

These pointer games may seem mysterious. In what sense does a higher-order program play such a game? A concrete explanation is given in (Levy, 2004a), using a language and a style of operational semantics that are more explicit about interaction between parts of programs (see also (Danos et al., 1996)). 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:

- Whenever it is his turn, Player can opt to state some $o \in O$ instead of an R-element
- 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 10 Let R be an arena, and let Z be an I/O signature.

(1) A *justified sequence* s in R wrt Z is a function from an initial[4] segment $\mathsf{moveset} \subseteq \mathbb{N}$ (whose elements are called *moves*) to

$$(\{*\} + \mathsf{moveset}) \times R + O + \textstyle\sum_{o \in O} I_o$$

such that, for each move m,
- $m \in I_o$ iff $m > 0$ and $s_{m-1} = o \in O$
- if $s_m = (*, r)$ then r is a root of R
- if $s_m = (n, r)$ where $n \in \mathsf{moveset}$, then $n < m$ and $s_n = (k, r')$ and r is a child of r'.

A move is described as an *arena move*, an *o-output move* or an *o-input move* according as s_m is (k, r) or o or $i \in I_o$. If s_m is (k, r) we say that "m points to k", and that m is *described as* $k \curvearrowleft r$.

[4] It is often convenient to generalize this so that moveset can be *any subset* of $\mathbb{N}$. It will then, of course, be uniquely order-isomorphic to an initial segment of $\mathbb{N}$, and by applying this order-isomorphism, we recover a "correct" justified sequence.

(2) A *play* is a justified sequence s such that for every move m,
- if m is even (e.g. 0) then it is either an output move or an arena move pointing to $*$ or to an odd arena move
- if m is odd then it is either an input move or an arena move pointing to an even arena move.

We then say that an arena move m is a *Player-move* or an *Opponent-move* according as m is even (e.g. 0) or odd.

(3) A finite play *awaits o-input* if it ends in an o-move. Otherwise it *awaits Player* or *awaits Opponent* according as its length is even or odd.

(4) An *nondeterministic infinite trace* (NIT) *strategy* σ for an arena R consists of
- a set A of Opponent-awaiting and input-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.

(5) We define *determinism*, and *deadlock-freedom* wrt a set U, for such strategies as in Def. 3.

(6) We define $\bigcup_{i\in I} \sigma_i$ and $\mathsf{input}^o\{\sigma_i\}_{i\in I_o}$ as in Def. 4.

(7) We write $\mathsf{strat}_{UZ}\ R$ for the set of strategies on R wrt Z, deadlock-free wrt U. This is clearly functorial in $R \in \mathbf{TokRen}$.

□

4.2 Categorical Structure

Fix an I/O signature Z and a set U of omni-errors. Our aim in this section is to define

- a category $\mathcal{G}_{UZ}$, with finite products
- a left $\mathcal{G}_{UZ}$-module, i.e. a functor $\mathcal{N}_{UZ} : \mathcal{G}_{UZ}^{\mathsf{op}} \to \mathbf{Set}$

(We often omit the subscripts UZ on $\mathcal{G}$ and $\mathcal{N}$ and strat). From this, together with some additional structure, we obtain our model of call-by-push-value. See (Levy, 2005) for a categorical account of this construction.

The objects of $\mathcal{G}$ are arenas, and $\mathcal{N}R$ is $\mathsf{strat}\ R$. We shall write $R \xrightarrow{\sigma}\!\!\!\!\!\rightarrow$ to mean $\sigma \in \mathsf{strat}\ R$. The homsets of $\mathcal{G}$ are given by

$$\mathcal{G}(R,S) = \prod_{b\in \mathsf{rt}\ S} \mathsf{strat}\ (R \uplus S{\restriction_b})$$

In fact, $\mathcal{G}$, (with determinism and bracketing constraints, and no I/O) is called the "thread-independence" category in (Abramsky et al., 1998)

To define the identity morphism on 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

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

Then $\mathsf{id}_R \in \mathcal{G}(R,R)$ is defined to map $b \in \mathsf{rt}\ R$ to $\mathsf{id}_{R,b}$.

To complete the above categorical structure, we need two kinds of composition: $R \xrightarrow{f} S \xrightarrow{g} T$ and $R \xrightarrow{f} S \xrightarrow{g}$ Both of these can be defined once we have, for arenas R, S, T, a map

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

which we are now going to define. Intuitively, the strategy $\sigma \circledast \tau$ 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 11 Let R,S,T be arenas.

(1) Let s be a justified sequence on $R \uplus S \uplus T$. The *inner thread-names* of such an s are

$$\mathsf{inners\ s} = \{\mathsf{left}\ am | a \in \mathsf{rt}\ S, m \text{ is a move in } s \text{ playing } * \curvearrowleft a\} \cup \{\mathsf{right}\}$$

The *thread-names of* s are $\mathsf{inners\ s} \cup \{\mathsf{outer}\}$. For each thread-name p, we define the *arena of* p to be
- $R \uplus T$ if $p = \mathsf{outer}$
- $S \uplus T$ if $p = \mathsf{right}$
- $R \uplus S{\restriction_a}$ if $p = \mathsf{left}\ am$.

Every thread-name other than outer is *inner*.

(2) A *collection of thread-pointers* for s associates to each rootmove in R an earlier rootmove in S, and to each output move an inner thread-name.

(3) Let s be an interaction pre-sequence on R, S, T, equipped with a collection of thread-pointers, and let q be a thread-name. We define the q-thread of s, a justified sequence on the arena of q, as follows.
- If $q = \mathsf{outer}$, it consists of all moves in R and T, and all output and input moves
- If q is inner, it consists of all output moves thread-pointing to q, all input moves that follow such an output moves, and
  - all moves in S and T , if $q = \mathsf{right}$
  - all R moves descended from a rootmove thread-pointing to m and all S moves *strictly* descended from m, if $q = \mathsf{left}\ am$.

s is an *interaction sequence* when all its threads (outer and inner) are plays.

(4) In an interaction sequence s, a thread-name q is *flashing* when
- $q = \mathsf{outer}$, and the outer-thread awaits Opponent, or
- q is inner, and the q-thread awaits Player.

□

Proposition 11 Let s be a finite interaction sequence on R, S, T.

- s has precisely one thread-name that is flashing, call it q.
- If sm is an interaction sequence, then m is in the q-thread of sm, and so q is not flashing in sm.
- If s has q-thread t, and tm is a play (in the arena of q), then s has a unique one-place extension whose q-thread is tm. (We shall write this sm, ignoring the reindexing of moves.)

If s is an infinite interaction sequence, then no thread-name is flashing. Therefore, an interaction sequence may be

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

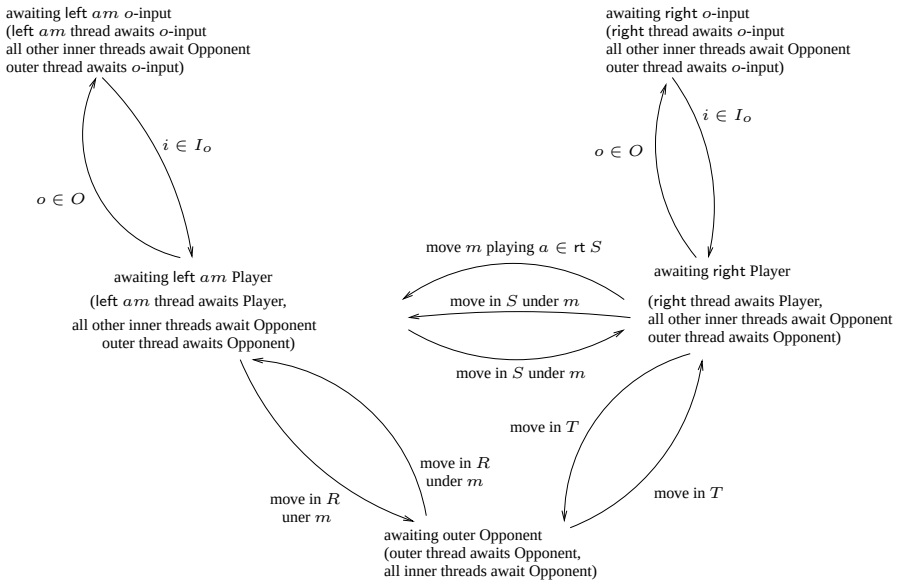
l-inner-Player awaiting finite, with outer-thread and l-inner thread awaiting Player, and all other inner threads awaiting Opponent

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.

□

The finite plays follow the state diagram



Using our classification of interaction sequences, we can now define the $\circledast$ operation.

Definition 12 Let R, S, T be arenas, and let $\sigma \in \mathcal{G}(R, S)$ and $\tau \in \mathsf{strat}\ (S \uplus T)$

(1) Let s be an interaction sequence on R, S, T. If q is a inner thread-name in s, we write $q(\sigma, \tau)$ to mean τ or σ_a according as q is right or left am. We say that s is *consistent* with σ and τ when
   if s **awaits outer-Opponent or** l**-input** for every inner thread-name q, each q-inner thread is a finite trace of $q(\sigma, \tau)$
   if s **awaits** l**-Player** the l-inner thread is a divergence of $l(\sigma, \tau)$, and for every inner thread-name $q \neq l$, each q-inner thread is a finite trace of $q(\sigma, \tau)$
   if s **is infinite** for every inner thread-name q, each q-inner thread is a finite trace of $q(\sigma, \tau)$

(2) We define $\sigma \circledast \tau$ to be
   finite traces the outer-thread of every outer-Opponent-awaiting or lo-input-awaiting interaction sequence s whose q-inner-thread is a finite trace of $q(\sigma, \tau)$ for every $q \in \mathsf{inners\ s}$
   divergences (1) the outer-thread of every l-Player awaiting interaction sequence s whose l-inner thread is a divergence of $l(\sigma, \tau)$ and whose q-inner thread is a finite trace of $q(\sigma, \tau)$ for every $q \in \mathsf{inners\ s} \backslash \{l\}$
   divergences (2) the outer-thread of every outer-starved interaction sequence whose q-inner-thread is a finite trace or infinite trace of $q(\sigma, \tau)$ for every $q \in \mathsf{inners\ s}$
   infinite traces the outer-thread of every outer-infinite interaction sequence consistent with σ and τ.

□

Proposition 12 The operation $\circledast$ preserves determinism, and deadlock-freedom wrt U. □

Using the $\circledast$ operation, we can define the two kinds of composition needed for our categorical structure.

Definition 13 (1) Given $\mathcal{G}_{UZ}$-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 $\sigma \circledast \tau_b$.

(2) Given $\mathcal{G}_{UZ}$-morphism $R \xrightarrow{\sigma} S$ and $S \xrightarrow{\tau}$, we define the composite $R \xrightarrow{\sigma;\tau}$ to be $\sigma \circledast \tau$ (taking T to be the empty arena).

□

Proposition 13 Def. 13(1) satisfies associativity and identity laws, making $\mathcal{G}$ a category. Def. 13(2) satisfies associativity and left-identity laws, making $\mathcal{N}_{UZ}$ a left $\mathcal{G}$-module. □

We define an identity-on-objects functor $\mathcal{F} : \mathbf{TokRen}^{\mathrm{op}} \longrightarrow \mathcal{G}$, taking f to the deterministic strategy given by token-renaming copycat.

Proposition 14 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 obtained by token-renaming along f. □

It follows immediately that $\mathcal{G}$ has finite products given by disjoint union, and that $\mathcal{F}$ preserves finite products on the nose.

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

Proposition 15 If $R \xrightarrow{\sigma} S$ and $R \uplus T \xrightarrow{\tau}$, then $\sigma \divideontimes \tau = (\sigma \times T); \tau$ □

4.3 Returning

In order to give the semantics of `return` V, we shall need to be able to convert a morphism $R \xrightarrow{\sigma} S\restriction_b$ into a strategy on $R \uplus S$, which we call $\mathsf{ret}(b, \sigma)$. This operation is called *returning*. Intuitively, Player begins with $* \curvearrowleft b$, and then every time Opponent points to this initial move, playing $0 \curvearrowleft c$, the Player begins a thread that follows σ_c.

The definition of returning is organized in a similar way to that of the $\divideontimes$ operation.

Definition 14 Let R and S be arenas, and let $b \in \mathsf{rt}\ S$.

(1) A *return pre-sequence* on R, S, b is an interaction sequence on $R \uplus S$ that begins with a b-rootmove m_0, and has no other rootmoves in S.
(2) The *inner thread-names* of such an s are

$$\mathsf{inners\ s} = \{\mathsf{ret}\ am | a \in \mathsf{children}(b), m \text{ plays } m_0 \curvearrowleft a\}$$

(3) The *thread-names* of such an s are

$$\mathsf{threads}\ s = \{\mathsf{outer}\} + \{\text{moves in } s \text{ pointing to the initial move}\}$$

The *thread-names of* s are $\mathsf{inners\ s} \cup \{\mathsf{outer}\}$. The *arena* of a thread-name is
- $R \uplus S$ if $q = \mathsf{outer}$
- $R \uplus S_a$ if $q = \mathsf{ret}\ am$.

Every thread-name other than outer is *inner*.
(4) A *collection of thread-pointers* for s associates to each output move an earlier inner thread-name.
(5) Let s be a return pre-sequence on R, S, T, equipped with a collection of thread-pointers, and let q be a thread-name. We define the q-thread of s, a justified sequence on the arena of q, as follows.
- If $q = \mathsf{outer}$, it consists of all the moves in s

- If $q = \mathsf{ret}\ am$, it consists of all R moves, all S-moves strictly descended from m, all output moves threadpointing to m and all input moves that follow such output moves.

s is a *return sequence* when, for every thread-name q, the q-thread is a play.

(6) In a return sequence s, a thread-name q is *flashing* when
- $q = \mathsf{outer}$, and the outer-thread awaits Opponent, or
- $q = \mathsf{ret}\ am$, and the q-thread awaits Player.

□

Proposition 16 Let s be a finite return sequence on R, S, b.

- s has precisely one thread-name that is flashing, call it q.
- If sm is a return sequence, then m is in the q-thread of sm, and so q is not flashing in sm.
- If s has q-thread t, and tm is a play (in the arena of q), then s has a unique one-place extension whose q-thread is tm. (We shall write this sm, ignoring the reindexing of moves.)

If s is an infinite interaction sequence, then no thread-name is flashing. Therefore, a return 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-infinite infinite, with outer-thread infinite, and each inner thread awaiting Opponent or infinite.

□

Using this classification of return sequences, we can define the returning operation.

Definition 15 Let R and S be arenas, let $b \in \mathsf{rt}\ S$, and let $R \xrightarrow{\sigma} S{\restriction_b}$.

(1) Let s be a return sequence on R, S, b For each inner thread-name $q = \mathsf{ret}\ am$, we write $q(\sigma)$ to mean σ_a. We say that s is *consistent* with b and σ when
if s **awaits outer-Opponent** for every inner thread-name q, the q-thread is a finite trace of $q(\sigma)$
if s **awaits** l**-inner-Player** the l-thread is a divergence of $l(\sigma)$ and for every inner thread-name $q \neq l$, the q-thread is a finite trace of $q(\sigma)$
if s **is outer-infinite** for every inner thread-name q, the q-thread is a finite trace or infinite trace of $q(\sigma)$.

(2) We define $\mathsf{ret}(b, \sigma)$ to be the strategy on $R \uplus S$ defined as follows:
finite traces the outer-thread of every outer-Opponent awaiting return sequence s consistent with b and σ

divergences the outer-thread of every l-inner-Player awaiting return sequence s consistent with b and σ

infinite traces the outer-thread of every outer-infinite return sequence consistent with b and σ.

□

We can recover this operation from the categorical structure, as follows.

Proposition 17 Let R and S be arenas, let $b \in \mathsf{rt}\ S$, and let $R \xrightarrow{\sigma} S\restriction_b$. Then

$$\mathsf{ret}(b, \sigma) = \sigma \mathbin{⋇} (f..\mathsf{id}_{S,b})$$

where we write $S\restriction_b \uplus S \xrightarrow{f} S \uplus S\restriction_b$ for the obvious token-change. □

We could have taken this as the definition of returning. But instead we have given a direct definition, in order to make the semantics of the `return` construct more intuitive.

Lemma 2 If $\mathsf{ret}(b, \sigma) = \mathsf{ret}(b', \sigma')$ then $b = b'$ and $\sigma = \sigma'$ □

Proof Firstly, b' is a finite trace of $\mathsf{ret}(b, \sigma)$ iff $b' = b$. Secondly, let c be a child of b, and let s be a play on $R \uplus S_c$. Then s is a finite trace / divergence / infinite trace of σ iff bcs is a finite trace / divergence / infinite trace of $\mathsf{ret}(b, \sigma)$ □

4.4 Model of Call-By-Push-Value

Definition 16 (1) A Q/A-*labelled arena* is an arena R, with every element classified as *question* or *answer*, where every child of an anwser is a question.. It is Q-*rooted* when, moreover, every root is a question.

(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 ith root (which we call $\mathsf{root}\ i$). 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 $r \in R$, both r and all its ancestors are elements of S, with the same labelling and parent-child relationship.

(4) We write $\mathcal{E}_{\mathsf{c}}$ 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 $R_i \sqsubseteq S_i$. We write $\mathcal{E}_{\mathsf{v}}$ for the (non-small) cpo of countable families of Q-rooted Q/A-labelled arenas.

□

Our intention is that

- a closed value type should denote an element of $\mathcal{E}_{\mathsf{v}}$
- a closed computation type should denote an element of $\mathcal{E}_{\mathsf{c}}$.

More generally

- a value type with m free value type identifiers and n free computation type identifiers denotes a continuous function

$$\mathcal{E}_{\mathsf{v}}^m \times \mathcal{E}_{\mathsf{c}}^n \longrightarrow \mathcal{E}_{\mathsf{v}}$$

- a computation type with m free value type identifiers and n free computation type identifiers denotes a continuous function

$$\mathcal{E}_{\mathsf{v}}^m \times \mathcal{E}_{\mathsf{c}}^n \longrightarrow \mathcal{E}_{\mathsf{c}}$$

with type recursion (of either kind) interpreted as least fixpoint. The semantics of the various connectives are given as follows:

$$\begin{aligned}
U\{R_i\}_{i\in I} &= \{\mathsf{pt}^{\mathsf{Q}}_{i\in I} R_i\}_{()\in 1} \\
\{R_i\}_{i\in I} \times \{S_j\}_{j\in J} &= \{R_i \uplus S_j\}_{(i,j)\in I\times J} \\
\textstyle\sum_{i\in I}\{R_{ij}\}_{j\in J_i} &= \{R_{ij}\}_{(i,j)\in \sum_{i\in I} J_i} \\
F\{R_i\}_{i\in I} &= \{\mathsf{pt}^{\mathsf{A}}_{i\in I} R_i\}_{()\in 1} \\
\{R_i\}_{i\in I} \to \{S_j\}_{j\in J} &= \{R_i \uplus S_j\}_{(i,j)\in I\times J} \\
\textstyle\prod_{i\in I}\{R_{ij}\}_{j\in J_i} &= \{R_{ij}\}_{(i,j)\in \sum_{i\in I} J_i}
\end{aligned}$$

A context Γ denotes an element of $\mathcal{E}_{\mathsf{v}}$, calculated using the $\times$ operation.

The semantics of judgements is as follows. Suppose Γ denotes $\{R_i\}_{i\in I}$.

- If the type $\underline{B}$ denotes $\{S_j\}_{j\in J}$, then a computation or a configuration inhabiting $\Gamma \vdash B$ denotes an element of

$$[\![\Gamma \vdash^{\mathsf{c}} \underline{B}]\!]_{UZ} = \prod_{i\in I}\prod_{j\in J} \mathsf{strat}_{UZ}\,(R \uplus S_j)$$

- If the type B denotes $\{S_j\}_{j\in J}$, then a value inhabiting $\Gamma \vdash^{\mathsf{v}} B$ denotes an element of

$$[\![\Gamma \vdash^{\mathsf{v}} B]\!]_{UZ} = \prod_{i\in I}\sum_{j\in J} \mathcal{G}_{UZ}(R_i, S_j)$$

- If the type $\underline{B}$ denotes $\{S_j\}_{j\in J}$ and the type $\underline{C}$ denotes $\{T_k\}_{k\in K}$, then a stack inhabiting $\Gamma|\underline{B} \vdash^{\mathsf{k}} \underline{C}$ denotes an element of

$$[\![\Gamma|\underline{B} \vdash^{\mathsf{k}} \underline{C}]\!]_{UZ} = \prod_{i\in I}\prod_{k\in K}\sum_{j\in J} \mathcal{G}_{UZ}(R_i \uplus T_k, S_j)$$

Semantics of terms is as follows. Let Γ denote $\{R_i\}_{i\in I}$, and write $\cdots\!\succ$ for token changing.

- $\texttt{choose}^h$ and $\texttt{input}^o$ are interpreted by $\bigcup$ and input^o.
- The operations of projection, λ, tupling, `fold` , `unfold` and stacking applications are interpreted by token-changing.
- Suppose A denotes $\{S_j\}_{j\in J}$. Then $\Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} \mathtt{x} : A$ at $i \in I, j \in J$ denotes j together with the $\mathcal{G}_{UZ}$-morphism $R_i \uplus S_j \longrightarrow S_j$ given at $b \in \mathsf{rt}\ S_j$ by

$$1 \xrightarrow{\mathsf{id}_{\hat{S},j}} \mathsf{strat}\ (S_j \uplus S_j\restriction_b) \dashrightarrow \mathsf{strat}\ ((R_i \uplus S_j) \uplus S_j\restriction_b)$$

  Other identifiers and `nil` are interpreted similarly.
- Suppose A denotes $\{S_j\}_{j\in J}$. If $\Gamma \vdash^{\mathsf{v}} V : A$ at $i \in I$ denotes $j \in J$ together with the $\mathcal{G}$-morphism $R_i \xrightarrow{\sigma} S_j$, then `return` V at $i, ()$ denotes

$$\begin{array}{rl} \mathcal{G}(R_i, S_j) & \dashrightarrow \mathcal{G}(R_i, \mathsf{pt}^{\mathsf{A}}_{j\in J} S_j\restriction_{\mathsf{root}\ j}) \\ & \Big\downarrow{\scriptstyle \mathsf{ret}(\mathsf{root}\ j, -)} \\ & \mathsf{strat}\ (R_i \uplus \mathsf{pt}^{\mathsf{A}}_{j\in J} S_j) \end{array}$$

  applied to σ.
- Suppose A denotes $\{S_j\}_{j\in J}$ and $\underline{B}$ denotes $\{T_k\}_{k\in K}$. Suppose $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}$, and $i \in I$ and $k \in K$. Suppose V at i denotes $j \in J$ together with the $\mathcal{G}$-morphism $R_i \xrightarrow{\sigma} S_j$, and M at $i, (j, k)$ denotes $R_i \uplus (S_j \uplus T_k) \xrightarrow{\tau}$. Then $V \text{`} M$ at i, k denotes

$$\begin{array}{c} \mathcal{G}(R_i, S_j) \times \mathsf{strat}\ (R_i \uplus (S_j \uplus T_k)) \\ \Big\downarrow \\ \mathcal{G}(R_i, S_j) \times \mathsf{strat}\ (S_j \uplus (R_i \uplus T_k)) \\ \Big\downarrow{\scriptstyle *} \\ \mathsf{strat}\ (R_i \uplus (R_i \uplus T_k)) \\ \Big\downarrow \\ \mathsf{strat}\ (R_i \uplus T_k) \end{array}$$

  applied to σ, τ. The operations of `let`, `pm`, stacking of sequencing contexts, and forming a configuration from a computation and a stack are all interpreted similarly.

The easy properties are as follows.

Proposition 18 (1) If the erratic signature Y is empty, then the denotation of every term, stack and configuration d is deterministic.

(2) [soundness] For any configuration d, we have $[\![d]\!] =$
- $[\![M]\!]$ if $d = \Gamma, M, \underline{C}, \mathtt{nil}, \underline{C}$
- $\bigcup_{d \rightsquigarrow d'} [\![d']\!]$, if d' is silent
- $\mathsf{input}^o\{[\![d : i]\!]\}_{i\in I_o}$ if d is an o-state

□

Proof (1) follows from Prop. 12. For (17), we note that all the term constructs are defined in terms of the $\divideontimes$ operation, the ret operation and token renaming, and each of these has been characterized in terms of the categorical structure (Prop. 15, Prop. 17, and Prop. 4.2 respectively). This enables us to first prove a substitution lemma, and then deduce (2) from the categorical structure. □

To state adequacy, we require the following definition:

Definition 17 Let Y be an erratic signature deadlock-free wrt U, and Z an I/O signature. For a configuration d in $\mathcal{L}^{\mathsf{CBPV}}(Y, Z, \Gamma \vdash^{\mathsf{c}} \underline{B})$, where Γ denotes R and C denotes $\{S_j\}_{j\in J}$, we write $[\![d]\!]$ for the element of $[\![\Gamma \vdash^{\mathsf{c}} \underline{B}]\!]_{UZ}$ that maps $j \in J$ 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 $[\![T]\!]j$.

□

Proposition 19 Let d be a configuration in $\mathcal{L}^{\mathsf{CBPV}}(Y, Z, \Gamma \vdash^{\mathsf{c}} \underline{C})$, where Y is deadlock-free.

(1) If the erratic signature Y is empty, then $[\![d]\!]$ is deterministic.
adequacy $[d] = [\![d]\!]$.

□

Prop. 19(1) follows from Prop. 18(1) and the determinism of the BLTS $\mathcal{L}^{\mathsf{CBPV}}(Y, Z, \Gamma \vdash C)$. Proving Prop. 19(1) is the task of Sect. 4.6.

Corollary 20 If d is a configuration inhabiting $\vdash^{\mathsf{c}} F\sum_{i\in I} 1$ then $[d] = [\![d]\!]$. □

Proof It is evident that $[\![d]\!] = [d]$. □

4.5 Hiding

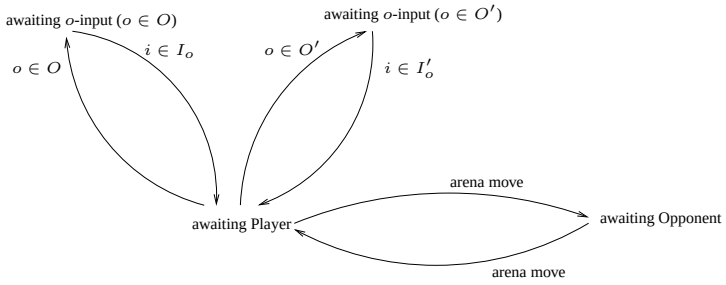
Before we prove computational adequacy, we adapt our study of hiding in Sect. 2.5 to strategies for pointer games. Let $Z = \{I_o\}_{o\in O}$ and $Z' = \{I'_o\}_{o\in O'}$ be I/O signatures.

Proposition 21 (cf. Prop. 7) Let s be a play wrt $Z + Z'$ into V. Write $s \setminus Z'$ for the play wrt Z into V obtained by suppressing all I/O moves in Z'. Then precisely one of the following hold:

awaiting outer input s and $s \setminus Z'$ await o-input, where $o \in Z$
awaiting outer Opponent s and $s \setminus Z'$ await Opponent
awaiting inner Player s and $s \setminus Z'$ await Player
awaiting hidden input s awaits o-input, where $o \in Z'$, and $s \setminus Z'$ awaits Player
outer starved s is infinite and $s \setminus Z'$ awaits Player.
outer infinite s and $s \setminus Z'$ are infinite

(Here, "outer" refers to $s \setminus Z'$ and "inner" refers to s.) □

This is proved by induction for finite plays, which follow the state diagram



and is then trivial for the infinite plays.

Definition 18 (cf. Prop. 7) Given a strategy $\sigma = (A, B, C, D)$ into V wrt $Z + Z'$, the *hiding* of σ, written $\sigma \setminus Z'$, is the strategy wrt Z defined as follows

finite traces $s \setminus Z'$, where s awaits outer input or outer Opponent and is a finite trace of σ
divergences (1) $s \setminus Z'$, where s awaits inner Player and is a divergence of σ
divergences (2) $s \setminus Z'$, where s is outer starved and is an infinite trace of σ
infinite traces $s \setminus Z'$, where s is outer infinite and is an infinite trace of σ

□

The following result will be useful in our adequacy proof (Sect. 4.6). It is the analogue of Prop. 9 in the setting of strategies on arenas.

Proposition 22 Given signatures Z and Z', the hiding of

$$\bigcup_{i\in I}\sigma_i \text{ is } \bigcup_{i\in I}(\sigma_i \setminus Z')$$

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

$$\mathsf{id}_{R,b} \text{ is } \mathsf{id}_{R,b}$$

$$f..\sigma \text{ is } f..(\sigma \setminus Z')$$

$$\sigma \circledast \tau \text{ is } (\sigma \setminus Z') \circledast (\tau \setminus Z')$$

$$\mathsf{ret}(b,\sigma) \text{ is } \mathsf{ret}(b, \sigma \setminus Z')$$

where σ and τ and all σ_i are strategies wrt $Z + Z'$. □

Proof Most of these are trivial. The result that $\circledast$ commutes with hiding is proved by a "zipping" argument, similar to that used to prove associativity of composition. Here it is in detail.

Let R, S, T be arenas and let Z and Z' be I/O signatures. Define A to be the pointed cpo of plays over Z on $R \uplus T$, ordered by extension. Define B to be the pointed cpo of interaction sequences over $Z + Z'$ on R, S, T, ordered by extension. Define C to be the poset of pairs (u, v), where

- u is an interaction sequence over Z on R, S, T
- v associates, to each inner thread-index q in u, a play $v(q)$ over $Z + Z'$ on the arena of q such that $v(q) \setminus Z'$ is the q-thread of u.

The ordering makes $(u, v) \leqslant (u', v')$ when u is a prefix of u' and, for every inner thread-name q in u, the play $v(q)$ is a prefix of the play $v'(q)$. We note that, in C, if $(u, v) < (u', v')$ then precisely one of the following hold.

(1) u awaits outer-arena-Opponent; then for each inner thread-name q, the play $v(q) \setminus Z'$ awaits arena-Opponent and so $v(q)$ awaits arena-Opponent. Hence $u < u'$ (because $v(q) < v'(q)$ implies $u < u'$).

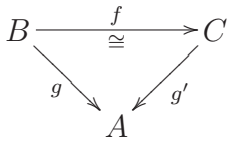
(2) u awaits o-input (where $o \in Z$) in thread l; then the play $v(l) \setminus Z'$ awaits o-input, so $v(l)$ awaits o-input; and for each inner thread-name $q \neq l$, the play $v(q) \setminus Z'$ awaits arena-Opponent so $v(q)$ awaits arena-Opponent. Hence $u < u'$ (because $v(q) < v'(q)$ implies $u < u'$).

(3) u is infinite; then for each inner thread-name q, the play $v(q) \setminus Z'$ awaits arena-Opponent and so $v(q)$ awaits arena-Opponent. Hence $u < u'$ (because $v(q) < v'(q)$ implies $u < u'$)—impossible.

(4) u awaits l-inner-Player; then for each inner thread-name $q \neq l$, the play $v(l) \setminus Z'$ awaits Player and so $v(q)$ awaits arena-Opponent or Z-input. Hence $v(l) < v'(l)$ (because $u < u'$ implies $v(l) < v(l')$, and $v(q) < v'(q)$ implies $u < u'$ if $q \neq l$). So $v(l)$ is finite, and since $v(l) \setminus Z'$ awaits Player, $v(l)$ either awaits Player or awaits Z'-input.

In particular, we see that u and every $v(q)$ must be finite. So every element of C with infinitely many predecessors is maximal.

We construct a commutative diagram:



Here,

- g maps an interaction sequence s to its outer thread restricted to Z.
- g' maps (u, v) to the outer thread of u
- f maps an interaction sequence s to $(s \setminus Z', v)$, where $v(q)$ is the q-inner thread of v (using the correspondence between the S-rootmoves in s and the S-rootmoves in $s \setminus Z'$).

Clearly these are strict continuous maps and clearly the diagram commutes. The function f is strictly monotone, because every move in s appears somewhere in $f(s)$. We show that if $s \in B$ and $(u', v') \in C$ and $f(s) < (u', v')$, then the set $\{t \in B | s < t, f(t) \leqslant (u', v')\}$ has a least element s', by an extensive case analysis.

For example: if $f(s) = (u, v)$ is of the form (1), then s is awaiting outer-arena-Opponent. We know that $um \leqslant u'$, and m plays $n \curvearrowleft r$. Suppose $r \in R$. Then n is a Player-move in some thread l. Hence $(v(l) \setminus Z')(n \curvearrowleft r) \sqsubseteq v'(l) \setminus Z'$, so $v(l)(n \curvearrowleft r) \sqsubseteq v(l)$. Put $s' = s(n \curvearrowleft r)$; then $f(s') = (u'', v'')$ where $u'' = u(n \curvearrowleft r)$ and $v''(l) = v(l)(n \curvearrowleft r)$ and $v''(q) = v(q)$ for every inner thread-name $q \neq l$. Hence $f(s') \leqslant (u', v')$, as required. If $f(sm') \leqslant (u', v')$, then m' must appear in u', so must be $n \curvearrowleft r$. Hence s' is the least element of $\{t \in B | s < t, f(t) \leqslant (u', v')\}$. The case where $r \in T$, and all the other cases, are similar.

For an element (u, v) of C, define the maximal sequence

$$s_0 < s_1 < s_2 < \cdots \in B \tag{2}$$

such that s_i is the unique B-element of length i whose f-image is $\leqslant (u, v)$. This is defined by induction: $\perp = s_0$, and if $f(s_i) < (u, v)$ then s_{i+1} is the least element of $\{t \in B | s_i < t, f(t) \leqslant (u, v)\}$. If (2) ends in s_n, then $f(s_n) = (u, v)$. If (2) is infinite, set s_∞ to be $\bigsqcup_{i \in \mathbb{N}} s_i$, then $f(s_\infty) \leqslant (u, v)$. But $f(s_\infty)$ has infinitely many predecessors, so it is maximal. Thus, in either case, we have s such that $f(s) = (u, v)$. If $f(s') = (u, v)$, then every finite prefix of s' appears in (2), so $s' \leqslant s$, and, since f is strictly monotonic, $s' = s$. Thus f is an isomorphism.

Now suppose we are given $R \xrightarrow{\sigma} S$ and $S \uplus T \xrightarrow{\tau}$ over $Z + Z'$. Let t be a Player-awaiting play over Z on $R \uplus T$. Then t is a divergence of $(\sigma \circledast \tau) \setminus Z'$ are iff $t = g'(s)$, for some $s \in B$ such that (condition 1)

- s awaits l-Player, its l-thread is a divergence of $l(\sigma, \tau)$ and every inner thread $q(\neq l)$ is a finite trace of $q(\sigma, \tau)$, or
- s is infinite, and every inner thread q is a finite trace or infinite trace of $q(\sigma, \tau)$

And t is a divergence of $(\sigma \setminus Z') \circledast (\tau \setminus Z')$ iff $t = g'(u, v)$, for some $(u, v) \in C$ such that (condition 2)

- u awaits l-Player, $v(l)$ is a divergence of $l(\sigma, \tau)$, and for $q \neq l$, $q(l)$ is a finite trace of $q(\sigma, \tau)$
- u awaits l-Player, $v(l)$ is an infinite trace of $l(\sigma, \tau)$, and for $q \neq l$, $q(l)$ is a finite trace of $q(\sigma, \tau)$
- u is infinite, and each $v(q)$ is a finite trace or infinite trace of $q(\sigma, \tau)$

Any $s \in B$ satisfies condition 1 iff $f(s)$ satisfies condition 2, so $(\sigma \circledast \tau) \setminus Z'$ and $(\sigma \setminus Z') \circledast (\tau \setminus Z')$ have the same divergences. By a similar but easier argument, they have the same finite traces and infinite traces. □

Finally, Prop. 10, stating that every strategy is obtainable by taking a deterministic strategy and hiding some of the I/O operators, is equally true here, with the same proof. A consequence of this, taken together with Prop. 22, is that all the equations between strategies stated in Sect. 4.2, as well as Prop. 17, can be deduced for all NIT strategies once we know that they are true for deterministic strategies.

4.6 Proving Computational Adequacy

To prove Prop. 19(1), our basic plan is this. We take the BLTS over Z for our language, and apply to it the *unhiding* operation as described in the proof of Prop. 10(1). This gives a BLTS over $Z + Z'$ that is deterministic and has no divergences. We then give a denotational semantics in deterministic strategies over $Z + Z'$. Because both operational and denotational semantics are deterministic, and the operational semantics has no divergences, it is easy to deduce adequacy from soundness. Now if we hide Z' in $[\![d]\!]_{Z+Z'}$, we get back $[\![d]\!]_Z$—mainly because hiding commutes with composition. Likewise, if we hide Z' in $[\![\![d]\!]\!]_{Z+Z'}$, we get back $[\![\![d]\!]\!]_{Z+Z'}$. So we can deduce the adequacy of $[\![-]\!]_Z$ from that of $[\![-]\!]_{Z+Z'}$.

To save us the work of defining $[\![-]\!]_{Z+Z'}$, and proving another soundness theorem, we do not use hiding and unhiding on BLTS's. Instead, we simulate unhiding with an *unhiding transform* that does two things:

- after each step of execution, print a tick
- turn each erratic choice into requested input

Thus, the transform of a configuration d, written $\overline{d}$, is deterministic (because it contains no erratic choice) and cannot diverge (because each step is observable). It

is easy to prove adequacy for such a term. Now if we take the denotation of $\overline{d}$, and hide both the $\checkmark$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 $\overline{d}$.

Although we cannot yet prove Prop. 19(1), we can deduce a weak form of it from Prop. 18(2):

Lemma 3 Let d inhabit $\Gamma \vdash C$, where $[\![C]\!] = \{S_j\}_{j \in J}$. Suppose $j \in J$.

(1) 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 $[\![d]\!]j$.
(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 define the unhiding transform from $\mathcal{L}^{\mathsf{CBPV}}(Y, Z)$ to $\mathcal{L}^{\mathsf{CBPV}}(\{\}, Z + (Y + \{\checkmark\}))$ in Fig. 5. Essentially, subterms that are computations acquire a tick.

The following lemma gives the operational properties of the unhiding transform; it does not mention the game semantics at all.

Lemma 4 (1) If $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$ and $\Gamma \vdash^{\mathsf{v}} V : A$ then $\overline{M[V/\mathtt{x}]} = \overline{M}[\overline{V}/\mathtt{x}]$.
(2) Let $d = \Gamma, M, \underline{B}, K, \underline{C}$. If M is not `choose` or `input`, then either
- d and $\overline{d}$ are terminal, or
- d is silent with unique successor d', and $\overline{d}$ is silent with unique successor $\checkmark.\overline{d'}$.

(3) $[\overline{d}]$ has no divergences.
(4) If $[d] = (A, B, C, D)$ then $[\overline{d}] \setminus Z' = (A, B, C, \{s\overline{T} | sT \in D\})$

□

We can prove adequacy for $\overline{d}$ i.e.

$$[\![\overline{d}]\!] = \overline{[\![\overline{d}]\!]} \tag{3}$$

because the LHS is deterministic (Prop. 18(1)), the RHS is deterministic (Prop. 19(1)) and they have the same finite traces (Lemma 3(2)–(3) and Lemma 4(3)).

We wish to deduce Prop. 19(1) from (3). Prop. 22 tells us that for any term, stack or configuration P, we have $\overline{[\![\overline{P}]\!] \setminus Z'} = [\![P]\!]$. Hence, Lemma 4(fitem:oprvl) tells us that for any configuration d, we have $\overline{[\![\overline{d}]\!] \setminus Z'} = [\![d]\!]$. We conclude

$$[\![d]\!] = [\![\overline{d}]\!] \setminus Z' = \overline{[\![\overline{d}]\!]} \setminus Z' = [\![d]\!]$$

$\Gamma \vdash^{\mathsf{c}} M : \underline{B}$	$\Gamma \vdash^{\mathsf{c}} \overline{M} : \underline{B}$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ N$	$\mathtt{let}\ \overline{V}\ \mathtt{be}\ \mathtt{x}.\ \checkmark.\overline{N}$
$\mathtt{return}\ V$	$\mathtt{return}\ \overline{V}$
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\checkmark.\overline{M}\ \mathtt{to}\ \mathtt{x}.\ \checkmark.\overline{N}$
$\mathtt{force}\ V$	$\mathtt{force}\ \overline{V}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i\in I}$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\checkmark.\overline{M_i}\}_{i\in I}$
$\mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\checkmark.\overline{M}$
$\lambda\{i.M_i\}_{i\in I}$	$\lambda\{i.\checkmark.\overline{M_i}\}_{i\in I}$
$\hat{\imath}\text{`}M$	$\hat{\imath}\text{`}\checkmark.\overline{M}$
$\lambda\mathtt{x}.M$	$\lambda\mathtt{x}.\checkmark.\overline{M}$
$V\text{`}M$	$\overline{V}\text{`}\checkmark.\overline{M}$
$\mathtt{pm}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.M$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\checkmark.\overline{M}$
$\mathtt{fold}\ M$	$\mathtt{fold}\ \checkmark.\overline{M}$
$\mathtt{unfold}\ M$	$\mathtt{unfold}\ \checkmark.\overline{M}$
$\mathtt{choose}^h\{M_p\}_{p\in P_h}$	$\mathtt{input}^h\{\overline{M_p}\}_{p\in P_h}$
$\mathtt{input}^o\{M_i\}_{i\in I_o}$	$\mathtt{input}^o\{\overline{M_i}\}_{i\in I_o}$

$\Gamma \vdash^{\mathsf{v}} V : B$	$\Gamma \vdash^{\mathsf{v}} \overline{V} : B$
$\mathtt{x}$	$\mathtt{x}$
$\langle \hat{\imath}, V\rangle$	$\langle \hat{\imath}, \overline{V}\rangle$
$\langle V, V'\rangle$	$\langle \overline{V}, \overline{V'}\rangle$
$\mathtt{thunk}\ M$	$\mathtt{thunk}\ \checkmark.\overline{M}$
$\mathtt{fold}\ V$	$\mathtt{fold}\ \overline{V}$

$\Gamma \vert \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$	$\Gamma \vert \underline{B} \vdash^{\mathsf{k}} \overline{K} : \underline{B}$
$\mathtt{nil}$	$\mathtt{nil}$
$\hat{\imath} :: K$	$\hat{\imath} :: \overline{K}$
$V :: K$	$\overline{V} :: \overline{K}$
$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ M :: K$	$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ \checkmark.\overline{M} :: \overline{K}$
$\mathtt{unfold} :: K$	$\mathtt{unfold} :: \overline{K}$

d inhabits $\Gamma \vdash^{\mathsf{c}} \underline{C}$	$\overline{d}$ inhabits $\Gamma \vdash^{\mathsf{c}} \underline{C}$
$\Gamma, M, \underline{B}, K, \underline{C}$	$\Gamma, \overline{M}, \underline{B}, \overline{K}, \underline{C}$

Fig. 5. The Unhiding Transform

as required.

5 Definability And Full Abstraction

In this section, we explain how to reduce the definability and full abstraction questions for NIT strategies to a definability property for deterministic strategies.

Definition 19 (1) We write detstrat R for the set of deterministic strategies on R, over the empty I/O signature.

(2) If Γ denotes $\{R_i\}_{i\in I}$ and $\underline{B}$ denotes $\{S_j\}_{j\in J}$, we define

$$[\![\Gamma \vdash^{\mathsf{c}} \underline{B}]\!]_{\mathsf{det}} \text{ to be } \prod_{i\in I, j\in J} \mathsf{detstrat}\ (R_i \uplus S_j)$$

(3) An extension $\mathcal{L}'$ of CBPV is *deterministic strategy complete* when every $\sigma \in [\![\Gamma \vdash^{\mathsf{c}} \underline{B}]\!]_{\mathsf{det}}$ is the denotation of some $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ in $\mathcal{L}'$. (Note that σ might

not be computable.)

(4) If Y is an erratic signature deadlock-free wrt U and Z is an I/O signature, we write $\mathcal{L}'(Y, Z)$ to mean $\mathcal{L}'$ extended with erratic choice and I/O from these signatures.

□

We do not discuss in this paper how deterministic strategy completeness may be achieved: it requires both control operators and general references. See (Abramsky et al., 1998; Laird, 1998) for more details. For the rest of the section we assume that $\mathcal{L}'$ is deterministic strategy complete.

Surprisingly, the definability and full abstraction properties appear to be mutually exclusive. Definability requires an erratic choice operator with arity much bigger than the I/O signature. Full abstraction, on the other hand, requires an I/O signature much bigger than the arities of the erratic choice operators. There does not appear to be any pair of erratic and I/O signatures making both properties true.

Proposition 23 (definability) Let Z be an I/O signature and let U be a set of omni-errors. There exists an erratic signature Y, deadlock-free wrt U, such the model of $\mathcal{L}'(Y', Z)$ is *junk-free* in the following sense. Every $\sigma \in [\![\Gamma \vdash^{\mathsf{c}} M : \underline{B}]\!]_{UZ}$ is the denotation of some computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ in $\mathcal{L}'(Y, Z)$. □

Proof (similar to that of Prop. 10)

If U is nonempty Define Y to consist of one operator h of arity $P_h = 2^{\mathsf{max}(\aleph_0, |Z|)}$ and one nullary operator h'. Let Γ denote $\{R_i\}_{i \in I}$ and let $\underline{B}$ denote $\{S_j\}_{j \in J}$. If $i \in I, j \in J$ and s an Opponent-awaiting play on $R_i \uplus S_j$, let τ_{ijs} be the deterministic strategy on $R_i \uplus \mathsf{Q} \uplus S_j$ starting from s given by

$$(\{s\mathsf{Q}(m\mathsf{Q})^*\}, \{\}, \{s\mathsf{Q}(m\mathsf{Q})^\omega\})$$

Since I and each R_i and S_j are countable, there exists $U \subseteq P_h$ and a surjection $U \xrightarrow{f} \sum_{i \in I, j \in J}(A_{ij} \cup B_{ij} \cup C_{ij})$. For each $u \in P_h$, define an element $g(u)$ of

$$[\![\Gamma, \mathtt{x} : UF0 \in \underline{B}]\!]_{\mathsf{det}} = \prod_{i \in I, j \in J} \mathsf{detstrat}\,(R_i \uplus \mathsf{Q} \uplus S_j)$$

whose finite traces are

$$\{t | t \text{ awaits Opponent}, t \sqsubseteq s\} \cup$$
$$\{t |\ t \text{ awaits Opponent}, t \sqsubseteq s, tm \not\sqsubseteq s, tmu \text{ a finite trace of } \tau_{ijtm}\}$$

whose divergences are

$$\{s | s \text{ awaits Player}\} \cup$$
$$\{t |\ t \text{ awaits Opponent}, t \sqsubseteq s, tm \not\sqsubseteq s, tmu \text{ a divergence of } \tau_{ijtm}\}$$

and whose infinite traces are

$$\{s | s \text{ is infinite}\} \cup$$
$$\{t |\ t \text{ awaits Opponent}, t \sqsubseteq s, tm \not\sqsubseteq s, tmu \text{ an infinite trace of } \tau_{ijtm}\}$$

For each $u \in P_h$, by deterministic strategy completeness, there exists a computation $\Gamma, \mathtt{x} : UF0 \vdash^{\mathsf{c}} M : \underline{B}$ denoting $g(u)$. Then it is clear that the term

$$\Gamma \vdash^{\mathsf{c}} \mathtt{let\ thunk\ choose}^{h'}\{\}\ \mathtt{be\ x.\ choose}^{h}\{M_h\}_{u \in P_h} : \underline{B}$$

denotes σ.

If U is empty Define erratic signature Y to consist of just one operator h of arity $P_h = 2^{\mathsf{max}(\aleph_0, |Z|)}$. The rest is similar to the case of U being empty.

□

Denotational equality in the case of values and stacks can be reduced to the case of computations:

Proposition 24 (1) If $\Gamma \vdash^{\mathsf{v}} V, V' : B$ have distinct denotations, then so do

$$\Gamma \vdash^{\mathsf{c}} \mathtt{return}\, V,\ \mathtt{return}\, V' : FB$$

(2) If $\Gamma | \underline{B} \vdash^{\mathsf{k}} K, K' : \underline{C}$ have distinct denotations, then so do

$$\Gamma, \mathtt{x} : U\underline{B} \vdash^{\mathsf{c}} (\mathtt{force\ x}) \bullet K,\ (\mathtt{force\ x}) \bullet K' : \underline{C}$$

□

Proof (1) follows from Lemma 2. (2) is proved similarly. □

Proposition 25 (full abstraction) Let Z be an I/O signature, and let Y be an erratic signature deadlock-free wrt U. There exists an I/O signature $Z' \supseteq Z$ such that the model of $\mathcal{L}(Y, Z')$ is *fully abstract* in the following sense. If two computations $\Gamma \vdash^{\mathsf{c}} M, M' : \underline{B}$ have distinct denotations, then there exists a context $\mathcal{C}[\cdot] : F\sum_{i \in I} 1$ with hole in $\Gamma \vdash^{\mathsf{c}} \underline{B}$ in such that $[\mathcal{C}[M], \mathtt{nil}] \neq [\mathcal{C}[M'], \mathtt{nil}]$. □

Note that, by Prop. 24, it suffices to state this result for computations.

Proof We construct Z' by adding to Z a collection of $2^{\mathsf{max}\{\aleph_0, |Y|, |Z|\}}$ printing operators. For M, M' in $\mathcal{L}'(Y, Z')$ there must be a countable sequence $o_0, o_1, \ldots$ of

printing operators in Z' that do not appear in M, M'.

Consider the computation judgement $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{c}} \underline{B}$, where $\overrightarrow{\mathtt{x} : A}$ denotes $\{R_i\}_{i \in I}$ and $\underline{B}$ denotes $\{S_j\}_{j \in J}$. Let $i \in I, j \in J$ and let s be an Opponent-awaiting / Player-awaiting / infinite play over Z' on $R_i \uplus S_j$ that does not involve $o_0, o_1, \ldots$.

We claim that there exists

- a context $\mathcal{C}_{ijs}[\cdot] : F(1+1)$ with hole of type $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{c}} \underline{B}$
- an Opponent-awaiting / Player-awaiting / infinite play t_{ijs} over Z' on the arena $\mathsf{A}_{\mathsf{true}}\mathsf{A}_{\mathsf{false}}$

such that for every $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{c}} M'' : \underline{B} \in \mathcal{L}'(Y, Z')$ not mentioning $o_0, o_1, \ldots$, the play s is a finite trace / divergence / infinite trace of $[\![M'']\!]ij$ iff the play t_{ijs} is a finite trace / divergence / infinite trace of $[\![\mathcal{C}_{ijs}[M'']]\!]$, which by Cor. 20 is equal to $[\mathcal{C}[M'']\mathtt{nil}]$. This proves the required property.

To establish the claim, we first define an element σ_{ijs} of

$$[\![\mathtt{y} : U(\overrightarrow{A} \to \underline{B}), \mathtt{z} : U\textstyle\prod_{i \in \mathbb{N}} F1 \vdash^{\mathsf{c}} (1+1)]\!]_{\mathsf{det}} =$$

$$\mathsf{detstrat}\ (\mathsf{pt}^{\mathsf{Q}}_{i \in I, j \in J}(R_i \uplus S_j) \uplus \begin{matrix} \mathsf{Q}_0 & \mathsf{Q}_1 & \mathsf{Q}_2 \\ \mathsf{A}_0 & \mathsf{A}_1 & \mathsf{A}_2 \end{matrix} \cdots \uplus \mathsf{A}_{\mathsf{true}}\mathsf{A}_{\mathsf{false}})$$

and the play t over Z'.

Now s takes one of the following forms:

$$s_0 m_0 m_0' \cdots s_{k-1} m_{k-1} m_{k-1}' s_k m_k \tag{4}$$
$$s_0 m_0 m_0' \cdots s_{k-1} m_{k-1} m_{k-1}' s' \tag{5}$$
$$s_0 m_0 m_0' s_1 m_1 m_1' \cdots \tag{6}$$

where each s_i is an even-length sequence of I/O moves, and each m_i is an arena Player-move.

Suppose s is of the form (4). We define σ_{ijs} to be the deterministic strategy that has finite traces

$$\{\mathsf{Q}_{ij} m_0 \mathsf{Q}_0 \mathsf{A}_0 m_0' \cdots m_{l-1} \mathsf{Q}_{l-1} \mathsf{A}_{l-1} m_{l-1}' | l \leqslant k\} \cup$$
$$\{\mathsf{Q}_{ij} m_0 \mathsf{Q}_0 \mathsf{A}_0 m_0' \cdots m_{l-1} \mathsf{Q}_{l-1} \mathsf{A}_{l-1} m_{l-1}' m_l \mathsf{Q}_l | l < k\} \cup$$
$$\{\mathsf{Q}_{ij} m_0 \mathsf{Q}_0 \mathsf{A}_0 m_0' \cdots m_{l-1} \mathsf{Q}_{l-1} \mathsf{A}_{l-1} m_{l-1}' n \mathsf{A}_{\mathsf{false}} (m \mathsf{A}_{\mathsf{false}})^* | l \leqslant k, n \neq m_l\} \cup$$
$$\{\mathsf{Q}_{ij} m_0 \mathsf{Q}_0 \mathsf{A}_0 m_0' \cdots m_{k-1} \mathsf{Q}_{k-1} \mathsf{A}_{k-1} m_{k-1}' m_k \mathsf{A}_{\mathsf{true}} (m \mathsf{A}_{\mathsf{false}})^*\}$$

has no divergences, and has infinite traces

$$\{\mathsf{Q}_{ij} m_0 \mathsf{Q}_0 \mathsf{A}_0 m'_0 \cdots m_{l-1} \mathsf{Q}_{l-1} \mathsf{A}_{l-1} m'_{l-1} n \mathsf{A}_{\mathsf{false}} (m \mathsf{A}_{\mathsf{false}})^\omega | l \leqslant k, n \neq m_l\} \cup$$
$$\{\mathsf{Q}_{ij} m_0 \mathsf{Q}_0 \mathsf{A}_0 m'_0 \cdots m_{k-1} \mathsf{Q}_{k-1} \mathsf{A}_{k-1} m'_{k-1} m_k \mathsf{A}_{\mathsf{true}} (m \mathsf{A}_{\mathsf{false}})^\omega\}$$

and we define t_{ijs} to be $s_0 o_0() \cdots s_{k-1} o_{k-1}() s_k \mathsf{A}_{\mathsf{true}}$.

The definitions of σ_{ijs} and t_{ijs} in the case that s is of the form (5) or (6) are similar.

In each case we define $\mathtt{y} : U(\overrightarrow{A}, \mathtt{z} : U \prod_{i \in \mathbb{N}} F1 \vdash^{\mathsf{c}} N_{ijs} : F(1+1))$ to be a computation in $\mathcal{L}'$ that denotes σ. Then we define the context $\mathcal{C}_{ijs}[\cdot]$ to be

$$\mathtt{let\ thunk}\ \lambda \overrightarrow{\mathtt{x}}.[\cdot]\ \mathtt{be\ y.\ let\ thunk}\ \lambda\{i.\mathtt{input}^{o_i}\{\mathtt{return}\ ()\}_{()\in 1}\}_{i\in\mathbb{N}}\ \mathtt{be\ z.}\ N_{ijs}$$

and the claim is easily verified. □

6 Further Work

The adequacy proof above should be adapted to general references (Abramsky et al., 1998), but this seems likely to go through smoothly.

Two challenging problems remain in this area:

(1) characterizing those strategies definable with only countable choice
(2) characterizing those strategies definable without storage.

References

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

Brookes, S., 2002. The essence of Parallel Algol. Information and Computation 179.

Cattani, G. L., Winskel, G., 1997. Presheaf models for concurrency. In: Proc., 10th International Conference on Computer Science Logic, Utrecht, 1996. Vol. 1258 of LNCS.

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

Escardó, M., 1998. A metric model of PCF, unpublished research note.

Felleisen, M., Friedman, D., 1986. Control operators, the SECD-machine, and the λ-calculus. In: Wirsing, M. (Ed.), Formal Description of Programming Concepts III. North-Holland, pp. 193–217.

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

Hasegawa, M., 1997. Models of sharing graphs :–a categorical semantics of let and letrec. Ph.D. thesis, University of Edinburgh.

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

Jonsson, B., 1994. A fully abstract trace model for dataflow and asynchronous networks. Distributed Computing 7 (4), 197–212.

Laird, J., 1998. A semantic analysis of control. Ph.D. thesis, University of Edinburgh.

Levy, P. B., 1999. Call-by-push-value: a subsuming paradigm (extended abstract). In: Girard, J.-Y. (Ed.), Proceedings, Typed Lambda-Calculi and Applications, L'Aquila, Italy. Vol. 1581 of LNCS. Springer, pp. 228–242.

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

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

Levy, P. B., 2005. Adjunction models for call-by-push-value with stacks. Theory and Applications of Categories 14, 75–110.

Levy, P. B., 2006. Infinite trace equivalence. In: Proceedings, 21st Annual Conference in Mathematical Foundations of Computer Science, Birmingham, UK, 2005. No. 155 in ENTCS. pp. 467–496.

McCusker, G., 1996. Games and full abstraction for a functional metalanguage with recursive types. Ph.D. thesis, University of London.

Moggi, E., 1991. Notions of computation and monads. Information and Computation 93, 55–92.

Panangaden, P., Russell, J. R., 1989. A category-theoretic semantics for unbounded indeterminacy. In: Proc., 5th Conference on Mathematical Foundations of Programming Semantics, New Orleans. Vol. 442 of LNCS.

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

Plotkin, G., Power, J., 2002. Notions of computation determine monads. In: Proc., Foundations of Software Sci. and Comp. Struct., 2002. Vol. 2303 of LNCS. Springer.

Roscoe, A. W., 1998. Theory and Practice of Concurrency. Prentice-Hall.

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