Abstract

We solve a longstanding problem by providing a denotational model for nondeterministic programs that identifies two programs iff they have the same range of possible behaviours. We discuss the difficulties with traditional approaches, where divergence is bottom or where a term denotes a function from a set of environments. We see that making forcing explicit, in the manner of game semantics, allows us to avoid these problems.

We begin by modelling a first-order language with sequential I/O and unbounded nondeterminism (no harder to model, using this method, than finite nondeterminism). Then we extend the semantics to higher-order and recursive types by adapting earlier game models. Traditional adequacy proofs using logical relations are not applicable, so we use instead a novel hiding argument.

Keywords: nondeterminism, infinite traces, game semantics

1 Introduction

1.1 The Problem

Consider the following call-by-name\(^2\) language of countably nondeterministic commands:

\[ M ::= \ x \mid \text{print } c. \ M \mid \mu x. M \mid \text{choose } n \in \mathbb{N}. M_n \]

where \( c \) ranges over some alphabet \( \mathcal{A} \). We define binary nondeterminism \( M \) or \( M' \) from countable in the evident way.
A closed term can behave in two ways: to print finitely many characters and then diverge, or to print infinitely many characters. Two closed terms are said to be \textit{infinite trace equivalent} when they have the same range of possible behaviours.

As stated in [15], “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 [15] and divergence semantics [18], identify programs that are not infinite trace equivalent, so they are too coarse. Others count the internal manipulations [2,4] or include branching-time information, so they are too fine (at best) for this problem.

In this paper, we provide a solution, and see that it can be used to model not only the above language, but also unbounded nondeterminism, input (following a request), and higher-order, sum and recursive types. Our model is a form of pointer game semantics [8], although the technology of pointer games is needed only for the higher-order types. This gives a good illustration of the power and flexibility of game semantics.

Proving the computational adequacy of the model incorporating higher-order, sum and recursive types presents a difficulty, because the traditional method, using a logical relation, is not applicable to it. So we give, instead, a proof that uses the method of \textit{hiding}. As a byproduct, we obtain a very simple proof of the adequacy of the game model of FPC [13].

### 1.2 Why Explicit Forcing?

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

1. A \textit{divergence-least} semantics is one where a term denotes an element of a poset, every construct is monotone, and \([\mu x.x]\) denotes a least element \(\bot\). Examples are the Hoare, Smyth and Plotkin powerdomain semantics [15], all the CSP semantics in [17], and the game semantics of [6]. Divergence-least semantics cannot model infinite trace equivalence, by the following argument taken from [15]. (We abbreviate \texttt{print ✓} as ✓.)

   \[
   \begin{align*}
   M &= \bot \text{ or } ✓.✓.\bot \\
   M' &= \bot \text{ or } ✓.\bot \text{ or } ✓.✓.\bot \\
   \text{Then } M &= \bot \text{ or } ✓.✓.\bot \leq M' \\
   M &= \bot \text{ or } ✓.✓.\bot \text{ or } ✓.✓.\bot \geq M'
   \end{align*}
   \]

Hence \(M = M'\), contradicting infinite trace equivalence. This argument uses only binary nondeterminism.
(ii) A well-pointed semantics is one where (roughly speaking) a term denotes a function from the set of environments. Examples are the 3 powerdomain semantics [15], all the CSP semantics in [17], the semantics using infinite traces in [2], and divergence semantics [18]. In general, well-pointed semantics are appropriate for equivalences satisfying the context lemma property: terms equivalent in every environment are equivalent in every context. However, infinite trace equivalence does not satisfy this property, as the following two terms\(^3\) involving \(x\) demonstrate:

\[
N = (\text{choose } n. \check{n}. \mu z.z) \text{ or } x
\]
\[
N' = (\text{choose } n. \check{n}. \mu z.z) \text{ or } \check{x}
\]

On the one hand, \(N\) and \(N'\) are infinite trace equivalent in every environment, because \(\check{\cdot}\) is the only character:

<table>
<thead>
<tr>
<th>closed term</th>
<th>(N[M/x])</th>
<th>(N'[M/x])</th>
</tr>
</thead>
<tbody>
<tr>
<td>can print (\check{n}) then diverge</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>can print (\check{\omega})</td>
<td>iff (M) can print (\check{\omega})</td>
<td>iff (M) can print (\check{\omega})</td>
</tr>
</tbody>
</table>

On the other hand, they are not contextually equivalent:

<table>
<thead>
<tr>
<th>closed term</th>
<th>(\mu x. N)</th>
<th>(\mu x. N')</th>
</tr>
</thead>
<tbody>
<tr>
<td>can print (\check{n}) then diverge</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>can print (\check{\omega})</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

so any model of infinite trace equivalence must distinguish them.

(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 \(\text{choose}^{\bot} n\in\mathbb{N}. M_n\) to be \((\mu f \lambda n. (M_n \text{ or } f(n + 1)))0\), which either executes some \(M_n\) or diverges [15]. Using this instead of \(\text{choose} \ n\in\mathbb{N},\) the same problem arises.)

\(^3\) discovered by A. W. Roscoe in 1989 [personal communication], and independently in [10].
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) \( x \), whereas \( N \) is not:

<table>
<thead>
<tr>
<th>term involving ( x )</th>
<th>( N )</th>
<th>( N' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>can print ( \check n ) then diverge</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>can print ( \check \omega )</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>can force ( x )</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>can print ( \check ) then force ( x )</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>can print ( \check n+2 ) then force ( x )</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

And that gives our solution.

This idea, that a model should make explicit when a call-by-name program forces its (thunked) argument, is present (often implicitly) in game semantics, where (as argued in [11]) “asking a question” indicates forcing a thunk. That is why our solution fits into the game framework. However, the game models in the literature are divergence-least, and this property is exploited by adequacy proofs using logical relations. This is even true of the nondeterministic model of [6], where strategy sets are quotiented by the Egli-Milner preorder and so they become cpos. The novelty of this paper is that it avoids such quotienting.

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

\[
P = \lambda x.(\text{diverge or if } x \text{ then (if } x \text{ then true else true) else true})
\]

\[
P' = \lambda x.(\text{diverge or (if } x \text{ then diverge else true) else true})
\]

or if \( x \) then (if \( x \) then true else true) else true

of type \( \text{bool} \rightarrow \text{bool} \). In [6], 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

\[
C[\cdot] = [\cdot](\check . \text{true})
\]

Now \( C[P] \) may print a tick and then diverge, whereas \( C[P'] \) 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.

1.4 Related Work: Dataflow Networks

An infinite trace model for dataflow networks—including feedback, but not recursion—was presented in [9], and shown fully abstract. In the terminology of [7], it forms a cartesian-centre traced symmetric monoidal category.

Although it is shown in [7] that such a category, if centrally closed, can be converted into a model of recursion—in a certain sense—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 2.1 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 $A$—determines a language $\mathcal{L}(Y, A)$ in which for each $h \in H$, there is a construct $\text{choose}^h\{M_p\}_{p \in P_h}$ that
erratically chooses \( p \in P_h \) and then executes \( M_p \). The syntax is

\[
M ::= x \mid \text{print } c. M \mid \mu x. M \mid \text{choose}^h \{M_p\}_{p \in P_h}
\]

For each context \( \Gamma = x_0, \ldots, x_{n-1} \), 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

\[
\begin{align*}
\mu x. M &\xrightarrow{\tau} M[\mu x. M/x] \\
\text{choose}^h \{M_p\}_{p \in P_h} &\xrightarrow{\tau} M_{\hat{p}} \quad (\hat{p} \in P_h) \\
\text{print } c. M &\xrightarrow{c} M
\end{align*}
\]

Given a set \( U \) of omni-errors, we also write \( M \xrightarrow{U} \) 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 \( \text{choose}^h \{\} \) 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:

(i) print finitely many characters then diverge

(ii) print infinitely many characters

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

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

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

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

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

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

**Proof.** (i) Consider \( \checkmark.\text{choose}^h \{\} \) and \( \text{choose}^h \{\} \), where \( P_h \) is empty.

(ii) By deadlock-freeness, every finite path extends to a path that is either infinite or ends in an omni-error. \( \square \)

Prop. 2.2(ii) 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 \( \text{input}^o\{ M_i \}_{i \in I_o} \) that prints \( o \), then waits for the user to supply some \( i \in I_o \), and then executes \( M_i \). We say \( Z \) is countable when \( O \) and each \( I_o \) is countable.

Given a signature \( Z \), we write \( R_Z \) for the endofunctor on \( \text{Set} \) mapping \( X \) to \( \sum_{o \in O} X^{I_o} \). We then obtain a strong monad \( T_Z \) on \( \text{Set} \) (the free monad on \( R_Z \)) mapping \( A \) to \( \mu Y.(A + R_Z Y) \). This monad can be used, in the manner of \[14,16\], to model requested input. Note that this includes as special cases the monads designated in \[14\] as “interactive input”, “interactive output”, and “exceptions”. We accordingly regard each output \( c \in A \) as an element of \( O \) such that \( I_o = 1 \), and we regard \( \text{print } c. M \) as syntactic sugar \(^4\) for \( \text{input}^c\{ M_i \}_{i \in I} \).

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 \( \text{Set} \)).

**Definition 2.3 (BLTS)** Let \( Z = \{ I_o \}_{o \in O} \) and \( U \) a set of omni-errors.

(i) 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 \( S_{\text{sil}} \) and \( S_o \) for the set of silent states and of \( o \)-states, respectively

- a relation \( \rightsquigarrow \) from \( S_{\text{sil}} \) to \( S \) and a function \( S_o \times I_o \longrightarrow S \) for each \( o \in O \).

\(^4\) The reader may feel that there is a substantial difference between these two things, because \( \text{print } c. M \) prints \( c \) and immediately executes \( M \), whereas \( \text{input}^c\{ M_i \}_{i \in I} \) prints \( c \) and then waits for a response before continuing to execute \( M_i \), 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.
It is \textit{deadlock-free} if either every silent state has at least one successor, or \( U \) is nonempty.

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

(iii) A terminable BLTS, is \textit{deterministic} when \( U \) is empty and 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”.

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

(i) An \textit{play} wrt \( Z \) is a finite or infinite sequence \( o_0 i_0 o_1 i_1 \ldots \) where \( o_r \in O \) and \( i_r \in I_o \). It \textit{awaits Player} if of even length, and \textit{awaits o-input} if of odd length ending in \( o \). A \textit{play over }\( Z \text{ terminating in } V \text{ is a Player-awaiting play extended with an element of } V \).

(ii) A \textit{nondeterministic infinite trace} (NIT) \textit{strategy over }\( Z \text{ into } V \) is a tuple \( \sigma = (A, B, C, D) \) where

- \textbf{finite traces} \( A \) is a set of input-awaiting plays
- \textbf{divergences} \( B \) is a set of Player-awaiting plays
- \textbf{infinite traces} \( C \) is a set of infinite plays
- \textbf{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 \( A \).

(iii) A Player-awaiting play \( t \) is \textit{finitely consistent} with a NIT strategy \( \sigma \) when every input-awaiting prefix of \( t \) is a finite trace of \( \sigma \).

(iv) A NIT strategy \( \sigma = (A, B, C, D) \) is \textit{deterministic} when

- any Player-awaiting play \( t \) finitely consistent with \( \sigma \) 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 \( \sigma \) is in \( C \).

(v) Let \( s \) be a Player-awaiting play over \( Z \) into \( V \). A NIT strategy over \( Z \) into \( V \text{ starting from } s \) 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 \textit{determinism} for such strategies as in (iv).

(vi) (adapted from [17]) A NIT strategy \( \sigma \) into \( V \) is \textit{deadlock-free} wrt a set \( U \) when either \( U \) is nonempty or, for every Player-awaiting \( s' \) that is finitely consistent with \( \sigma \), there is a deterministic strategy \( \tau \) starting from \( s' \) such that \( \tau \subseteq \sigma \).

(vii) 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 $\textbf{Set}$.

Here are some operations on strategies.

**Definition 2.5**  
(i) For $d \in V$, we define $\eta_d$ (the monad’s unit at $d$) to be the deterministic strategy
\[
(\{\}, \{\}, \{\}, \{d\})
\]
(ii) 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
\[
\left(\bigcup_{i \in I} A_i, \bigcup_{i \in I} B_i, \bigcup_{i \in I} C_i, \bigcup_{i \in I} D_i\right)
\]
(iii) 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 $\text{input}^o \{\sigma_i\}_{i \in I_o}$ 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 2.6**  
(i) A deterministic strategy is deadlock-free wrt any set $U$.
(ii) $\text{input}^o$ preserves determinism, and preserves deadlock-freedom wrt any set $U$.
(iii) If $I$ or $U$ is nonempty, then $\bigcup_{i \in I}$ preserves deadlock-freedom wrt $U$.

**Definition 2.7** (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.

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

**Proposition 2.8** Let $d$ be a state in a terminable BLTS $\mathcal{L}$ over $Z$. 
If $L$ is deterministic, then so is $[d]$.

If $L$ is deadlock-free wrt $U$, then so is $[d]$.

**Proposition 2.9** Let $d$ be a state in a terminable BLTS over $Z$.

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

### 2.4 Operational and Denotational Semantics

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 $L(Y, Z)$ whose syntax is given by

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

For each context $\Gamma = x_0, \ldots, x_{n-1}$, we write $L(Y, Z, \Gamma)$ 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
- $\mu x.M$ is silent, and $\mu x.M \leadsto M[\mu x.M/x]$
- $\text{choose}^h\{M_p\}_{p \in P_h}$ is silent, and $\text{choose}^h\{M_p\}_{p \in P_h} \leadsto M_p$ for each $\hat{p} \in P_h$
- $\text{input}^o\{M_i\}_{i \in I_o}$ is an $o$-state, and $(\text{input}^o\{M_i\}_{i \in I_o}) : i = M_i$ for each $i \in I_o$

The following is trivial.

**Lemma 2.10** Suppose $\Gamma, x \vdash M$ and $\Gamma \vdash N$. Suppose that $M$ is not $x$.

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

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

**Proposition 2.11** In the language $L(Y, Z)$, we have

- (i) If $x \in \Gamma$, then $[\Gamma \vdash x] = \eta x$
- (ii) $[\Gamma \vdash \text{choose}^h\{M_p\}_{p \in P_h}] = \bigcup_{p \in P_h} [\Gamma \vdash M_p]$.
- (iii) $[\Gamma \vdash \text{input}^o\{\Gamma M_i\}_{i \in I_o}] = \text{input}^o\{[\Gamma \vdash M_o]\}_{i \in I_o}$
(iv) If $\Gamma, x \vdash M$ then

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

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

$$(\{l_0 \cdots l_{n-1} l_0 x \in D, \ldots, l_{n-1} x \in D, l_0 \in A\},$$

$$\{l_0 \cdots l_{n-1} l_0 x \in D, \ldots, l_{n-1} x \in D, l \in B\}$$

$$\cup \{l_0 \cdots l_{n-1} | l_0 x \in D, \ldots, l_{n-1} x \in D, x \in D\},$$

$$\{l_0 \cdots l_{n-1} l_0 x \in D, \ldots, l_{n-1} x \in D, l \in C\}$$

$$\cup \{l_0 l_1 \cdots | l_0 x \in D, l_1 x \in D, \ldots \text{ and } l_0 l_1 \cdots \text{ is infinite}\},$$

$$\{l_0 \cdots l_{n-1} l y | l_0 x \in D, \ldots, l_{n-1} x \in D, l y \in D, y \neq x\})$$

(v) If $\Gamma, x \vdash M$ and $\Gamma \vdash N$, then

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

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

$$(A \cup \{ll'o | l x \in D, l'o \in A'\},$$

$$B \cup \{ll'| l x \in D, l' \in B'\},$$

$$C \cup \{ll'| l x \in D, l' \in C'\},$$

$$\{ly | l y \in D, y \neq x\} \cup \{ll'y | l x \in D, l'y \in D'\})$$

Prop. 2.11 gives us a computationally adequate denotational semantics.

**Proposition 2.12** The operations $\mu$ and $*$ defined in Prop. 2.11 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 2.13** Let $L$ be a terminable BLTS over $Z + Z'$. The **hiding** of $Z'$ in $L$, written $L \upharpoonright Z$, is the BLTS wrt $Z$ obtained from $L$ 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 2.14 Let $s$ be a play wrt $Z + Z'$ into $V$. Write $s \upharpoonright Z$ 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 $s \upharpoonright Z$ await $o$-input, where $o \in Z$

awaiting inner Player $s$ and $s \upharpoonright Z$ await Player

awaiting hidden input $s$ awaits $o$-input, where $o \in Z'$, and $s \upharpoonright Z$ awaits Player

outer starved $s$ is infinite and $s \upharpoonright Z$ awaits Player.

outer infinite $s$ and $s \upharpoonright Z$ are infinite

outer terminating $s$ and $s \upharpoonright Z$ are terminating

(Here, “outer” refers to $s \upharpoonright Z$ 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 2.15 Given a strategy $\sigma = (A, B, C, D)$ into $V$ wrt $Z + Z'$, the hiding of $\sigma$, written $\sigma \upharpoonright Z$, is the strategy wrt $Z$ defined as follows

finite traces $s \upharpoonright Z$, where $s$ awaits outer input and is a finite trace of $\sigma$

divergences (1) $s \upharpoonright Z$, where $s$ awaits inner Player and is a divergence of $\sigma$

divergences (2) $s \upharpoonright Z$, where $s$ is outer starved and is an infinite trace of $\sigma$

infinite traces $s \upharpoonright Z$, where $s$ is outer infinite and is an infinite trace of $\sigma$

terminating traces $s \upharpoonright Z$, where $s$ is outer terminating and is a terminating trace of $\sigma$.

Proposition 2.16 If $d$ is a state in a BLTS $\mathcal{L}$ over $Z + Z'$ then

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

Proposition 2.17 Given signatures $Z$ and $Z'$, the hiding of
\[
\begin{align*}
\eta v & \text{ is } \eta v \\
\bigcup_{i \in I} \sigma_i & \text{ is } \bigcup_{i \in I} (\sigma_i \upharpoonright Z)
\end{align*}
\]
\[
\text{input}^o \{ \sigma_i \}_{i \in I_o} \text{ is } \begin{cases} 
\text{input}^o \{ (\sigma_i \upharpoonright Z) \}_{i \in I_o} \text{ if } o \in Z \\
\bigcup_{i \in I_o} (\sigma_i \upharpoonright Z) \text{ if } o \in Z'
\end{cases}
\]

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

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

**Proposition 2.18** Let \( Z \) be an I/O signature.

(i) For every terminable BLTS \( L \) over \( Z \), there exists an I/O signature \( Z' \) and a deterministic terminable BLTS \( L' \) over \( Z + Z' \) such that \( L = L' \upharpoonright Z \).

(ii) There exists an I/O signature \( Z' \) with the following property. For every strategy \( \sigma \) over \( Z \) into a countable set \( V \), deadlock-free wrt a set \( U \), there exists a deterministic strategy \( \tau \) over \( Z + Z' \) into \( V \) such that \( \tau \upharpoonright Z = \sigma \).

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

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

\[
\text{input}^{o'} \{ g(i) \}_{i \in I(o')}
\]

It is then clear that \( \tau \upharpoonright Z = \sigma \).

The operation used in (i) is called *unhiding*.

### 3 Call-By-Name FPC

Again let \( Y = ( \{ P_h \}_{h \in H}, U) \) be a deadlock-free erratic signature and let \( Z = \{ I_o \}_{o \in I} \) be an I/O signature. We now define a *higher-order* language
\[ \mathcal{L}^{\text{FPC}}(Y, Z) \] by straightforwardly combining \( \mathcal{L}(Y, Z) \) with “call-by-name FPC” from [13]. Its types are just as in [13], i.e.

\[
A ::= A + A \mid 0 \mid A \times A \mid 1 \mid A \to A \mid X \mid \mu X.A
\]

The terms are given (omitting the constructs for 0 and 1) by

\[
M ::= x \mid \text{inl } M \mid \text{inr } M \mid \pi M \mid \pi' M \\
\mid \lambda x.M \mid MM \mid \text{fold } M \mid \text{unfold } M \\
\mid \text{pm } M \text{ as } \{\text{inl } x.N, \text{inr } x.N'\} \mid (M, M) \\
\mid \text{choose}^h \{M_p\}_{p \in P_h} \mid \text{input}^o\{M_i\}_{i \in I_o}
\]

where \( \text{pm} \) stands for “pattern-match”. We define the typing judgement \( \Gamma \vdash M : B \) inductively in the standard way. As for \( \mathcal{L}(Y, Z) \), we can regard it as the fragment of \( \mathcal{L}^{\text{FPC}}(Y, Z) \) in which the sole type is 0.

We give CK-machine semantics [5] in Fig. 1. As we work on a term, we keep a stack of contexts, similar to an evaluation context. For example, to evaluate \( \text{pm } M \text{ as } \{\text{inl } x.N, \text{inr } x.N'\} \), we first need to evaluate \( M \), so the rest of the term—the context \( \text{pm } [\cdot] \text{ as } \{\text{inl } x.N, \text{inr } x.N'\} \)—is placed onto the stack. Later, when \( M \) has been evaluated, this context is removed from the stack and used.

We write \( \Gamma|B \vdash^k K : C \) to mean that \( K \) is a stack that can accompany a term of type \( B \) in context \( \Gamma \), in the course of evaluating a term of type \( C \) in context \( \Gamma \). This judgement is defined inductively in Fig. 2.

We define a configuration inhabiting \( \Gamma \vdash C \) to consist of a type \( B \), a term \( \Gamma \vdash M : B \) and a stack \( \Gamma|B \vdash^k K : C \). We write \( \mathcal{L}^{\text{FPC}}(Y, Z, \Gamma \vdash C) \) for the terminable BLTS over \( Z \) and \( U \) whose states are the configurations inhabiting \( \Gamma \vdash C \), and which is defined in Fig. 1. It is deadlock-free because \( Y \) is.

**Note** Formally, all terms are deemed to be explicitly typed. However, to reduce clutter, we have not actually written the types.

### 4 Pointer Games

#### 4.1 Pointer Game On Arena

We obtain our model of CBN FPC by taking the standard game semantics of [13]—omitting for simplicity, the constraints of innocence, visibility and bracketing, although the latter two could easily be incorporated—and remove the nondeterminism constraint in the manner of Def. 2.4(ii).

To make the semantics of sum types work smoothly\(^5\) we make two superficial changes in the presentation:

\(^5\) The formulation of call-by-name sum types in [1] is not robust, because it only works
Initial Configuration to execute $\Gamma \vdash N : C$

<table>
<thead>
<tr>
<th>$\Gamma$</th>
<th>$N$</th>
<th>$C$</th>
<th>nil</th>
<th>$C$</th>
</tr>
</thead>
</table>

Transitions (we omit $\text{inr}$ and $\pi'$)

<table>
<thead>
<tr>
<th>$\Gamma$</th>
<th>$\Gamma$ pm $M$ as ${\text{inl } x, \text{inr } x.N'}$</th>
<th>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</th>
<th>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</th>
<th>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>$A + A'$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ inl $P$</td>
<td>$B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B \times B'$</td>
<td>$\pi[] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $(N, N')$</td>
<td>$B \times B'$</td>
<td>$\pi[] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B \times B'$</td>
<td>$\pi[] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ unfold $M$</td>
<td>$B[\mu X.B/X]$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu M$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td></td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu M$</td>
<td>$B \times B'$</td>
<td>$\pi[] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\pi M$</td>
<td>$B$</td>
<td>$K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
<tr>
<td>$\Gamma$ $\lambda x.P$</td>
<td>$A \rightarrow B$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
<td>$\Gamma$ pm $[]$ as ${\text{inl } x, \text{inr } x.N'}$</td>
</tr>
<tr>
<td>$\Gamma$ $\mu X.B$</td>
<td>$\mu X.B$</td>
<td>$\text{unfold} [] :: K$</td>
<td>$C$</td>
<td>silent, $\rightsquigarrow$</td>
</tr>
</tbody>
</table>

Terminal Configurations (we omit $\text{inr}$)

<table>
<thead>
<tr>
<th>$\Gamma$</th>
<th>$\lambda x.P$</th>
<th>$A \rightarrow B$</th>
<th>nil</th>
<th>$A \rightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma$</td>
<td>$(N, N')$</td>
<td>$B \times B'$</td>
<td>nil</td>
<td>$B \times B'$</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>$\text{inl } M$</td>
<td>$A + A'$</td>
<td>nil</td>
<td>$A + A'$</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>$\text{fold } M$</td>
<td>$\mu X.B$</td>
<td>nil</td>
<td>$\mu X.B$</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>$x$</td>
<td>$B$</td>
<td>nil</td>
<td>$C$</td>
</tr>
</tbody>
</table>

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

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

As an example, the type $\text{bool} \rightarrow \text{bool}$ will denote a singleton family consisting when both players are required to follow the bracketing condition. The reorganization here avoids that problem.
\[ \Gamma | C \vdash k \text{nil} : C \quad \Gamma | B \times B' \vdash k \pi[\cdot] :: K : C \]

\[ \Gamma, x : A \vdash k \text{N} : B \quad \Gamma, x : A \vdash N' : B \quad \Gamma | B \vdash k K : C \]

\[ \Gamma | A + A' \vdash k \text{pm}[\cdot] \text{ as } \{\text{inl } x.N, \text{inr } x.N'\} :: K : C \]

\[ \Gamma | N : B \quad \Gamma | B \vdash k K : C \]

\[ \Gamma | A \rightarrow B \vdash k N :: K : C \quad \Gamma | B[\mu X.B/X] \vdash k K : C \]

\[ \Gamma | \mu X.B \vdash k \text{unfold}[\cdot] :: K : C \]

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

of the arena

![Diagram](image)

We begin with arenas.

**Definition 4.1**

(i) Let \( R \) be a set equipped with a relation \( \vdash \subseteq (\{\ast\} + \text{moveset}) \times \text{moveset} \). The *roots* of \( R \) are \( \text{rt } R = \{r|\ast \vdash r\} \text{ rt } R \subseteq R \), 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

\[ \ast \vdash r_0 \vdash \cdots \vdash r_n = r \]

(ii) 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.)

(iii) We write \( R \cup S \) for the disjoint union of \( R \) and \( S \).

(iv) If \( r \in R \), we write \( R|s \) for the arena of elements strictly descended from \( r \).

(v) A *token renaming* from arena \( R \) to arena \( S \) is a function \( R \overset{f}{\rightarrow} S \), such that, if \( b \in \text{rt } R \), then \( fb \in \text{rt } S \) and \( f \) restricts to an arena isomorphism \( R|_b \cong S|_{fb} \). We write \( \text{TokRen} \) 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 \( r \in \text{rt } R \), 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

\[
\begin{array}{c}
\text{Q} \\
\text{A (true)} & \text{A (false)} \\
\end{array}
\]

is

\[
\begin{array}{c}
PQ & OA \text{ (true)} & PQ & OA \text{ (true)} \\
\end{array}
\]

(2)

In fact, this play will distinguish the terms \( P \) and \( P' \) given in Sect.1.2. In the case of \( P' \), the denotation includes (2) 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 [11], using a language and a style of operational semantics that are more explicit about interaction between parts of programs (see also [3]). Since all this is orthogonal to the nondeterminism which is the subject of this paper, we omit it.

An I/O signature \( Z \) determines a variation on this game:

• 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 4.2** Let \( R \) be an arena, and let \( Z \) be an I/O signature.

(i) A justified sequence \( s \) in \( R \) wrt \( Z \) is a function from an initial\(^6\) segment \( \text{moveset} \subseteq \mathbb{N} \) (whose elements are called moves) to

\[
(\{\ast\} + \text{moveset}) \times R + O + \sum_{o \in O} I_o
\]

such that, for each move \( m \),

• \( m \in I_o \) iff \( m > 0 \) and \( s_{m-1} = o \in O \)

\(^6\) It is often convenient to generalize this so that \( \text{moveset} \) 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.
• if \( s_m = (\ast, r) \) then \( r \) is a root of \( R \)
• if \( s_m = (n, r) \) where \( n \in \text{moveset} \), 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 \( s_m = (k, r) \) or \( o \) or \( i \in I_o \). If \( s_m = (k, r) \) we say that “\( m \) points to \( k \)”, and that \( m \) is described as \( k \prec r \).

(ii) 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 \( \ast \) 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.

(iii) 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.

(iv) An nondeterministic infinite trace (NIT) strategy \( \sigma \) 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 \( A \).

(v) We define determinism, and deadlock-freedom wrt a set \( U \), for such strategies as in Def. 2.4.

(vi) We define \( \bigcup_{i \in I} \sigma_i \) and \( \text{input}^a \{\sigma_i\}_{i \in I_o} \) as in Def. 2.5.

(vii) We write \( \text{strat}^{UZ} R \) for the set of strategies on \( R \) wrt \( Z \), deadlock-free wrt \( U \). This is clearly functorial in \( R \in \text{TokRen} \).

### 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 \( \mathcal{N}_{UZ} : \mathcal{G}_{UZ}^\text{o} \rightarrow \text{Set} \)
(We often omit the subscripts \( UZ \) on \( \mathcal{G} \) and \( \mathcal{N} \) and \( \text{strat} \)). From this, together with some additional structure, we obtain our model of call-by-push-value. See [12] for a categorical account of this construction.
The objects of $\mathcal{G}$ are arenas, and $\mathcal{N}R$ is $\text{strat} R$. We shall write $R \xrightarrow{\sigma} S$ to mean $\sigma \in \text{strat} R$. The homsets of $\mathcal{G}$ are given by

$$\mathcal{G}(R, S) = \prod_{b \in \text{rt} S} \text{strat} (R \uplus S|_b)$$

In fact, $\mathcal{G}$, (with determinism and bracketing constraints, and no I/O) is called the “thread-independence” category in [1].

To define the identity morphism on $R$, we define $\text{id}_{R,b}$ to be the deterministic strategy on $R \uplus R|_b$ with no divergences, and whose finite/infinite traces are all plays in which Player initially plays $\ast \triangleleft \text{inl} b$, and responds to

- $0 \triangleleft \text{inl} b$ with $\ast \triangleleft \text{inr} b$
- $n + 1 \triangleleft \text{inl} b$ with $n \triangleleft \text{inr} b$
- $n + 1 \triangleleft \text{inr} b$ with $n \triangleleft \text{inl} b$

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

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

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

which we are now going to define. Intuitively, the strategy $\sigma \ast \tau$ should follow $\tau$ until that plays a root $b$ of $S$, then continue in $\sigma_b$, until that plays another move in $S$, then follow $\tau$ again, and so forth. But the moves in $S$ are hidden—“parallel composition with hiding”.

**Definition 4.3** Let $R, S, T$ be arenas.

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

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

The thread-names of $s$ are $\text{inners } s \cup \{ \text{outer} \}$. For each thread-name $p$, we define the arena of $p$ to be

- $R \uplus T$ if $p = \text{outer}$
- $S \uplus T$ if $p = \text{right}$
- $R \uplus S|_a$ if $p = \text{left } am$.

Every thread-name other than $\text{outer}$ is inner.

(ii) 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.
(iii) 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 $q = \text{outer}$, 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 $q = \text{right}$
  - all $R$ moves descended from a rootmove thread-pointing to $m$ and all $S$ moves strictly descended from $m$, if $q = \text{left am}$

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

(iv) In an interaction sequence $s$, a thread-name $q$ is flashing when

- $q = \text{outer}$, and the outer-thread awaits Opponent, or
- $q$ is inner, and the $q$-thread awaits Player.

Proposition 4.4 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 \(*\) operation.

**Definition 4.5** Let \( R, S, T \) be arenas, and let \( \sigma \in \mathcal{G}(R, S) \) and \( \tau \in \text{strat} \ (S \cup T) \)

(i) 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 \( \tau \) or \( \sigma_a \) according as \( q \) is right or left am. We say that \( s \) is consistent with \( \sigma \) and \( \tau \) 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 \( q(\sigma, \tau) \)

(ii) We define \( \sigma \star \tau \) to be

- **finite traces** the outer-thread of every outer-Opponent-awaiting or **\( l \)-input-awaiting** interaction sequence \( s \) whose \( q \)-inner-thread is a finite trace of \( q(\sigma, \tau) \) for every \( q \in \text{inners } s \)
- **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 \( q \in \text{inners } s \star \{l\} \)
- **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 \( q \in \text{inners } s \)

- **infinite traces** the outer-thread of every outer-infinite interaction se-
Proposition 4.6 The operation \( \ast \) preserves determinism, and deadlock-freedom \( \text{wrt} \ U \).

Using the \( \ast \) operation, we can define the two kinds of composition needed for our categorical structure.

Definition 4.7 (i) Given \( G_{UZ} \)-morphisms \( R \stackrel{\sigma}{\rightarrow} S \) and \( S \stackrel{\tau}{\rightarrow} T \), we define the composite \( R \stackrel{\sigma \ast \tau}{\rightarrow} T \) at \( b \in \text{rt} \; T \) to be \( \sigma \ast \tau_b \).

(ii) Given \( G_{UZ} \)-morphism \( R \stackrel{\sigma}{\rightarrow} S \) and \( S \stackrel{\tau}{\rightarrow} \), we define the composite \( R \stackrel{\sigma \ast \tau}{\rightarrow} \) to be \( \sigma \ast \tau \) (taking \( T \) to be the empty arena).

Proposition 4.8 Def. 4.7(i) satisfies associativity and identity laws, making \( G \) a category. Def. 4.7(ii) satisfies associativity and left-identity laws, making \( N_{UZ} \) a left \( G \)-module.

We define an identity-on-objects functor \( F : \text{TokRen}^\text{op} \rightarrow G \), taking \( f \) to the deterministic strategy given by token-renaming copycat.

Proposition 4.9 All compositions of the form

\[
R \stackrel{\mathcal{F} f}{\rightarrow} S \stackrel{\sigma}{\rightarrow} T, \quad R \stackrel{\mathcal{F} f}{\rightarrow} S \stackrel{\sigma}{\rightarrow}, \quad \text{or} \quad R \stackrel{\sigma}{\rightarrow} S \stackrel{\mathcal{F} f}{\rightarrow} T
\]

are obtained by token-renaming along \( f \).

It follows immediately that \( G \) has finite products given by disjoint union, and that \( F \) preserves finite products on the nose.

The operation \( \ast \) can be recovered from the categorical structure:

Proposition 4.10 If \( R \stackrel{\sigma}{\rightarrow} S \) and \( R \uplus T \stackrel{\tau}{\rightarrow} \), then \( \sigma \ast \tau = (\sigma \times T) ; \tau \)

4.3 Returning

In order to give the semantics of \( \text{inl} \) and \( \text{inr} \), we shall need to be able to convert a morphism \( R \stackrel{\sigma}{\rightarrow} S \upharpoonright b \) into a strategy on \( R \uplus S \), which we call \( \text{ret}(b, \sigma) \). This operation is called \textit{returning}. Intuitively, Player begins with \( \ast \leftarrow b \), and then every time Opponent points to this initial move, playing \( 0 \leftarrow c \), the Player begins a thread that follows \( \sigma_b \). We omit the detailed definition, as it is structured similarly to the definition of composition.

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

Proposition 4.11 Let \( R \) and \( S \) be arenas, let \( b \in \text{rt} \; S \), and let \( R \stackrel{\sigma}{\rightarrow} S \upharpoonright b \). Then

\[
\text{ret}(b, \sigma) = \sigma \ast (f..\text{id}_{S,b})
\]
where we write $S \uplus S \xrightarrow{f} S \uplus S \uplus S$ for the obvious token-change.

We could have taken this as the definition of returning. But the direct definition makes the semantics of the $\text{inl}$ and $\text{inr}$ constructs more intuitive.

### 4.4 Model of Call-By-Name FPC

**Definition 4.12** (i) A Q/A-labelled arena is an arena $R$, with every element classified as question or answer, where no answer enables an answer. It is Q-rooted when, moreover, every root is a question.

(ii) For a countable family of Q/A-labelled arenas $\{R_i\}_{i \in I}$, we write $\text{pt}^Q_{i \in I} R_i$ for the labelled arena with $I$ roots, each a question, and a copy of $R_i$ placed below the $i$th root (which we call root $i$). Similarly $\text{pt}^A_{i \in I} R_i$, provided that each $R_i$ is Q-rooted.

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

(iv) We write $\mathcal{E}$ for the (non-small) cpo of countable families of Q/A-labelled arenas. $\{R_i\}_{i \in I} \sqsubseteq \{S_j\}_{j \in J}$ when for every $i \in I$, we have $j \in J$ and $R_i \sqsubseteq S_i$.

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

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

Semantics of judgements:

- A context $\Gamma = x_0 : A_0, \ldots, x_n : A_n$, where $A_k$ denotes $\{R_{k i}\}_{i \in I_k}$, denotes the labelled arena $\text{pt}^Q_{i \in I_0} R_{0 i} \uplus \cdots \uplus \text{pt}^Q_{i \in I_{n-1}} R_{(n-1),i}$.
- If the context $\Gamma$ denotes $R$, and the type $B$ denotes $\{S_j\}_{j \in J}$, then a term or a configuration inhabiting $\Gamma \vdash B$ denotes an element of

$$[[\Gamma \vdash B]]_{UZ} = \prod_{j \in J} \text{strat}_{UZ} (R \uplus S_j)$$

- If the context $\Gamma$ denotes $R$ and $A$ denotes $\{S_j\}_{j \in J}$ and the type $B$ denotes
\{T_k\}_{k \in K}, then a stack \(\Gamma|B \vdash^k K : C\) denotes an element of
\[
[\Gamma|B \vdash C]_{UZ} = \prod_{k \in K} \sum_{j \in J} G_{UZ}(R \uplus T_k, S_j)
\]

Semantics of terms is as follows. Let \(\Gamma\) denote \(R\), and write \(\text{strat} \rightarrow \text{strat}\) for token changing.

- \text{choose}^h and \text{input}^o are interpreted by \(\bigcup\) and \text{input}^o.
- The operations of projection, pairing, \(\lambda\), fold, unfold and stacking application, projection and unfold contexts are interpreted by token-changing.
- Suppose \(A\) denotes \(\{S_j\}_{j \in J}\), and write \(\hat{S}\) for \(\text{pt}_{j \in J} S_j\). Then \(\Gamma, x : A \vdash x : A\) at \(j\) denotes
\[
1 \xrightarrow{\text{id}_{S_j}} \text{strat} \left(\hat{S} \uplus \hat{S}|_{\text{root}_j}\right) \xrightarrow{\text{strat}} \text{strat} \left((R \uplus \hat{S}) \uplus S_j\right)
\]

Other identifiers and \text{nil} are interpreted similarly.

- Suppose \(A\) denotes \(\{S_j\}_{j \in J}\) and \(A'\) denotes \(\{S'_j\}_{j \in J'}\). Write \(\overline{S}\) for \(\text{pt}^A\{\text{pt}_{i \in J}^R, \text{pt}_{j \in J}^Q S_j\}\). If \(\Gamma \vdash M : A\) then \text{inl} \(M\) at () denotes
\[
\prod_{j \in J} \text{strat} \left(R \uplus S_j\right) \xrightarrow{\text{strat}} G(R, \overline{S}|_{\text{root inl}()})
\]
\[
\text{ret}(\text{root inl}(), -) \xrightarrow{\text{strat}} \text{strat} \left(R \uplus \overline{S}\right)
\]

applied to \([M]\). And \text{inr} is interpreted similarly.

- Suppose \(A\) denotes \(\{S_j\}_{j \in J}\) and \(B\) denotes \(\{T_k\}_{k \in K}\), and write \(\hat{S}\) for \(\text{pt}_{j \in J}^Q S_j\). If \(\Gamma \vdash M : A\) and \(\Gamma \vdash N : A \rightarrow B\), then \(NM\) at \(k\) denotes
\[
\prod_{j \in J} \text{strat} \left(R \uplus S_j\right) \times \text{strat} \left(R \uplus (\hat{S} \uplus T_k)\right)
\]
\[
G(R, \hat{S}) \times \text{strat} \left(\hat{S} \uplus (R \uplus T)\right)
\]
\[
* \xrightarrow{\text{strat}} \text{strat} \left(R \uplus (R \uplus T)\right)
\]
\[
\text{strat} \left(R \uplus T\right)
\]

applied to \([M], [N]k\). The operations of pattern-match, stacking a pattern-match context and forming a configuration are interpreted similarly.
The easy properties are as follows.

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

(ii) **[Soundness]** For any configuration $d$, we have $[d] =$

- $[M]$ if $d = \Gamma, M, C, \text{nil}, C$
- $\bigcup_{d \leadsto d'} [d']$, if $d'$ is silent
- $\text{input}^o \{[d : i]\}_{i \in I_o}$ if $d$ is an o-state

**Proof.** (i) follows from Prop. 4.6. For (4.11), we note that all the term constructs are defined in terms of the $*$ operation, the ret operation and token renaming, and each of these has been characterized in terms of the categorical structure (Prop. 4.10, Prop. 4.11, and Prop. 4.2 respectively). This enables us to first prove a substitution lemma, and then deduce (ii) from the categorical structure.

To state adequacy, we require the following definition:

**Definition 4.14** Let $Y$ be an erratic signature deadlock-free wrt $U$, and $Z$ an I/O signature. For a configuration $d$ in $\mathcal{L}^{\text{FPC}}(Y, Z, \Gamma \vdash C)$, where $\Gamma$ denotes $R$ and $C$ denotes $\{S_j\}_{j \in J}$, we write $[d]$ for the element of $[\Gamma \vdash C]_UZ$ 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 4.15** Let $d$ be a configuration in $\mathcal{L}^{\text{FPC}}(Y, Z, \Gamma \vdash C)$, where $Y$ is deadlock-free.

(i) If the erratic signature $Y$ is empty, then $[d]$ is deterministic.

adequacy $[d] = [d]$.

Prop. 4.15(i) follows from Prop. 4.13(i) and the determinism of the BLTS $\mathcal{L}^{\text{FPC}}(Y, Z, \Gamma \vdash C)$. Proving Prop. 4.15(i) is the task of Sect. 4.6.

**Corollary 4.16** If $d$ is a configuration inhabiting $\vdash c \text{ bool}$ then $[d] = [d]$.

**Proof.** It is evident that $[d] = [d]$. \qed

### 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 4.17 (cf. Prop. 2.14) Let $s$ be a play wrt $Z + Z'$ into $V$. Write $s \upharpoonright Z$ 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 $s \upharpoonright Z$ await $o$-input, where $o \in Z$
- **awaiting outer Opponent** $s$ and $s \upharpoonright Z$ await Opponent
- **awaiting inner Player** $s$ and $s \upharpoonright Z$ await Player
- **awaiting hidden input** $s$ awaits $o$-input, where $o \in Z'$, and $s \upharpoonright Z$ awaits Player
- **outer starved** $s$ is infinite and $s \upharpoonright Z$ awaits Player.
- **outer infinite** $s$ and $s \upharpoonright Z$ are infinite

(Here, “outer” refers to $s \upharpoonright Z$ and “inner” refers to $s$.)

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

![State Diagram](image)

and is then trivial for the infinite plays.

Definition 4.18 (cf. Prop. 2.15) Given a strategy $\sigma = (A, B, C, D)$ into $V$ wrt $Z + Z'$, the hiding of $\sigma$, written $\sigma \upharpoonright Z$, is the strategy wrt $Z$ defined as follows

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

The following result will be useful in our adequacy proof (Sect. 4.6). It is the analogue of Prop. 2.17 in the setting of strategies on arenas.

Proposition 4.19 Given signatures $Z$ and $Z'$, the hiding of
\[
\bigcup_{i \in I} \sigma_i \text{ is } \bigcup_{i \in I} (\sigma_i \upharpoonright Z)
\]

\[
\text{input}^o \{\sigma_i\}_{i \in I_o} \text{ is } \begin{cases} 
\text{input}^o \{(\sigma_i \upharpoonright Z)\}_{i \in I_o} \text{ if } o \in Z \\
\bigcup_{i \in I_o} (\sigma_i \upharpoonright Z) \text{ if } o \in Z'
\end{cases}
\]

\[
id_{R,b} \text{ is } id_{R,b}
\]

\[
f_{..} \text{ is } f_{..}(\sigma \upharpoonright Z)
\]

\[
\sigma \ast \tau \text{ is } (\sigma \upharpoonright Z) \ast (\tau \upharpoonright Z)
\]

\[
\text{ret}(b, \sigma) \text{ is } \text{ret}(b, \sigma \upharpoonright Z)
\]

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

**Proof.** Most of these are trivial. The result that \(\ast\) commutes with hiding is proved by a “zipping” argument, similar to that used to prove associativity of composition.

Finally, Prop. 2.18, 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. 4.19, is that all the equations between strategies stated in Sect. 4.2, as well as Prop. 4.11, 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. 4.15(i), 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. 2.18(i). 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 \(\llbracket d \rrbracket_{Z + Z'}\), we get back \(\llbracket d \rrbracket_Z\)—mainly because hiding commutes with composition. Likewise, if we hide \(Z'\) in \(\llbracket d \rrbracket_{Z+Z'}\), we get back \(\llbracket d \rrbracket_{Z+Z'}\). So we can deduce the adequacy of \(\llbracket - \rrbracket_Z\) from that of \(\llbracket - \rrbracket_{Z+Z'}\).

To save us the work of defining \(\llbracket - \rrbracket_{Z+Z'}\), 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 $\check{\ }$’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 $\llbracket - \rrbracket$. So we deduce adequacy for $d$ from that of $\overline{d}$.

Although we cannot yet prove Prop. 4.15(i), we can deduce a weak form of it from Prop. 4.13(ii):

**Lemma 4.20** Let $d$ inhabit $\Gamma \vdash C$, where $[C] = \{S_j\}_{j \in J}$. Suppose $j \in J$.

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

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

(iii) Every finite trace (divergence, infinite trace) of $[d] j$ is either a finite trace (divergence, infinite trace) of $[\overline{d}] j$ or an extension of a divergence of $[d]$.

We next define the unhiding transform from $\mathcal{L}_{\text{FPC}}(Y, Z)$ to $\mathcal{L}_{\text{FPC}}(\{\}, Z + (Y + \{\check{\ }\}))$. The translation on terms, stacks and configurations is defined in Fig. 3. The placing of $\check{\ }$’s is motivated by the decomposition in [11]—thunked subterms do not acquire a $\check{\ }$.

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

**Lemma 4.21**

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

(ii) Let $d = \Gamma, M, B, K, C$. 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 $\check{\overline{d'}}$.

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

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

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

$$[\overline{d}] = [\overline{d}]$$

(3)

because the LHS is deterministic (Prop. 4.13(i)), the RHS is deterministic (Prop. 4.15(i)) and they have the same finite traces (Lemma 4.20(ii)–(iii) and Lemma 4.21(iii)).

We wish to deduce Prop. 4.15(i) from (3). Prop. 4.19 tells us that for any term, stack or configuration $P$, we have $[\overline{P}] | Z = [P]$. Hence, Lemma 4.21
\[
\begin{array}{|c|c|}
\hline
\Gamma \vdash M : B & \Gamma \vdash \overline{M} : B \\
\hline
x & x \\
\lambda x. M & \lambda x. \overline{\overline{M}} \\
MN & (\overline{\overline{M}})N \\
(M, M') & (\overline{\overline{M}}, \overline{\overline{M'}}) \\
\pi M & \pi \overline{\overline{M}} \\
inl M & inl \overline{\overline{M}} \\
\text{pm } M \text{ as } \{\text{inl } x.N, \text{inr } x.N'\} & \text{pm } \overline{\overline{M}} \text{ as } \{\text{inl } x.\overline{\overline{N}}, \text{inr } x.\overline{\overline{N'}}\} \\
\text{fold } M & \text{fold } \overline{\overline{M}} \\
\text{unfold } M & \text{unfold } \overline{\overline{M}} \\
\text{choose}^h\{M_p\}_{p \in P_h} & \text{input}^h\{M_p\}_{p \in P_h} \\
\text{input}^o\{M_i\}_{i \in I_o} & \text{input}^o\{\overline{\overline{M_i}}\}_{i \in I_o} \\
\hline
\Gamma | B \vdash^k K : C & \Gamma | B \vdash^k \overline{\overline{K}} : C \\
\hline
\text{nil} & \text{nil} \\
[.]N :: K & [.]\overline{\overline{N}} :: \overline{\overline{K}} \\
\pi[.] :: K & \pi[.] :: \overline{\overline{K}} \\
\text{pm } [.] \text{ as } \{\text{inl } x.N, \text{inr } x.N'\} :: K & \text{pm } [.] \text{ as } \{\text{inl } x.\overline{\overline{N}}, \text{inr } x.\overline{\overline{N'}}\} :: \overline{\overline{K}} \\
\text{unfold } [.] :: K & \text{unfold } [.] :: \overline{\overline{K}} \\
\hline
\end{array}
\]

\text{Fig. 3. The Unhiding Transform}

(fitem:oprvl) tells us that for any configuration } d, \text{ we have } \llbracket \overline{\overline{d}} \rrbracket | Z = \llbracket d \rrbracket. \text{ We conclude}

\[
\llbracket \overline{\overline{d}} \rrbracket | Z = \llbracket d \rrbracket | Z = \llbracket d \rrbracket
\]

as required.
5 Further Work

The adequacy proof above should be adapted to general references [1], and definability and full abstraction results formulated. It remains to characterize (i) strategies definable with only countable choice (ii) strategies definable without storage.

References

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