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 model to a calculus with higher-order and recursive types, by adapting standard game semantics. Traditional adequacy proofs using logical relations are not applicable, so we use instead a novel hiding and unhiding argument.

Key words: nondeterminism, infinite traces, game semantics, Jump-With-Argument

1 Introduction

1.1 The Problem

Consider the following call-by-name [1] language of countably nondeterministic commands with recursion:

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

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

$$\mathtt{div} \stackrel{\text{def}}{=} \mathtt{rec}\ \mathtt{x}.\ \mathtt{x} \qquad\qquad \mathtt{choose}^{\perp} n_{\in\mathbb{N}}.\ M_n \stackrel{\text{def}}{=} \mathtt{div}\ \mathtt{or}\ \mathtt{choose}\ n_{\in\mathbb{N}}.\ M_n$$

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 [2] 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 "Seeing Beyond Divergence" or SBD 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 (Abramsky, 1983; Cattani and Winskel, 2003; 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, interactive input, 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).

[2] Although this quotation appears within a discussion of a calculus *without* recursion, the point it makes is a general one.

1.2 Why Explicit Forcing?

Before turning to our solution, we consider two kinds of semantics that have been studied.

(1) A *divergence-least* semantics is one where a term denotes an element of a poset, every construct is monotone, and `div` denotes a least element $\bot$. Examples 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). Let us say that ♣ is an insult and ♡ is an apology. Put

$$M \stackrel{\text{def}}{=} \texttt{div or } (\texttt{print } ♣.\ \texttt{print } ♡.\ \texttt{div})$$
$$M' \stackrel{\text{def}}{=} \texttt{div or } (\texttt{print } ♣.\ \texttt{div}) \texttt{ or } (\texttt{print } ♣.\ \texttt{print } ♡.\ \texttt{div})$$

Then

$$M = \texttt{div or div or } (\texttt{print } ♣.\ \texttt{print } ♡.\ \texttt{div}) \leqslant M'$$
$$M = \texttt{div or } (\texttt{print } ♣.\ \texttt{print } ♡.\ \texttt{div}) \texttt{ or } (\texttt{print } ♣.\ \texttt{print } ♡.\ \texttt{div}) \geqslant M'$$

Hence $M = M'$, contradicting infinite trace equivalence. Moreover, 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 the SBD semantics presented in (Roscoe, 2004), which can.

(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 SBD 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 ♣, and consider the following two terms [3] involving `x`.

$$N \stackrel{\text{def}}{=} (\texttt{choose}^{\bot} n_{\in\mathbb{N}}.\ (\texttt{print } ♣.)^n\ \texttt{div}) \texttt{ or } \texttt{x}$$
$$N' \stackrel{\text{def}}{=} (\texttt{choose}^{\bot} n_{\in\mathbb{N}}.\ (\texttt{print } ♣.)^n\ \texttt{div}) \texttt{ or print } ♣.\ \texttt{x}$$

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

On the one hand, N and N' are infinite trace equivalent in every environment:

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

On the other hand, they are not contextually equivalent:

closed term	$\mathtt{rec\ x}.\ N$	$\mathtt{rec\ x}.\ N'$
can print $\clubsuit^n$ then diverge	yes	yes
can print $\clubsuit^\omega$	no	yes

and so any model of infinite trace equivalence must distinguish them. In particular, $\mathtt{rec\ x}.\ N$ must stop insulting the customer, but that is not the case for $\mathtt{rec\ x}.\ N'$. Thus a semantics that identifies N and N', such as cpo-enriched semantics (Abramsky, 1983) and SBD semantics (Broy, 1986; Roscoe, 2004), cannot verify infinite liveness.

(Lest the reader think unbounded nondeterminism is to blame, note that if we allow recursion over $\mathbb{N}$-indexed families of commands, we can express $\mathtt{choose}^{\perp} n_{\in\mathbb{N}}.M_n$ as $(\mathtt{rec\ f}\lambda n_{\in\mathbb{N}}.\ (M_n\ \mathtt{or\ f}(n+1)))0$. So finite nondeterminism suffices to make this example.)

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 $\clubsuit^n$ then diverge	yes	yes
can print $\clubsuit^\omega$	no	no
can force $\mathtt{x}$	yes	no
can print $\clubsuit$ then force $\mathtt{x}$	no	yes
can print $\clubsuit^{n+2}$ 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

$$P = \lambda \mathtt{x}.(\mathtt{div}\ \mathtt{or}\ \mathtt{if}\ \mathtt{x}\ \mathtt{then}\ (\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{true})\ \mathtt{else}\ \mathtt{true})$$
$$P' = \lambda \mathtt{x}.(\mathtt{div}\ \mathtt{or}\ (\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{div}\ \mathtt{else}\ \mathtt{true})\ \mathtt{or}\ \mathtt{if}\ \mathtt{x}\ \mathtt{then}\ (\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{true})\ \mathtt{else}\ \mathtt{true})$$

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

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

Now $\mathcal{C}[P]$ and $\mathcal{C}[P']$ may print $\clubsuit$ and then diverge, whereas $\mathcal{C}[P]$ cannot. Therefore, from the viewpoint of infinite trace equivalence, P and P' must have different denotations.

1.3 Structure Of Paper

We adapt 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.

Secondly, in Sect. 2.2, we add *interactive input*, which is one of the computational effects studied in (Moggi, 1991) and is illustrated in Fig. 1. This is where a program does not take input silently from a stream, but first prints a message requesting input, and then waits until it is supplied.

In Sect. 2.5, we give a denotational semantics for this language; no sophisticated game techniques are required at this stage.

The third adaptation, in Sect. 4, moves to a language with higher-order and recursive types. In (Levy, 2006b), this was done as an extension of the call-by-name language. But giving game semantics directly for a call-by-name calculus is complicated, so in this paper we use the calculus that (as argued in (Levy, 2004a)) makes game semantics easiest: Jump-With-Argument (JWA), a continuation passing calculus. The game semantics of (Abramsky et al., 1998; Hyland and Ong, 2000; Nickau, 1996) is presented for JWA in (Levy, 2005); in this paper we merely adapt that model to include nondeterminism, interactive input and infinite trace equivalence.

A program in BASIC

```
10 INPUT "Hello. Enter your name (a string):" name$
20 INPUT "Enter your age (an integer):" age
30 IF age >= 18 THEN INPUT "Enter your address (a string):" d$
```

A nondeterministic program—states marked "•" make an erratic choice

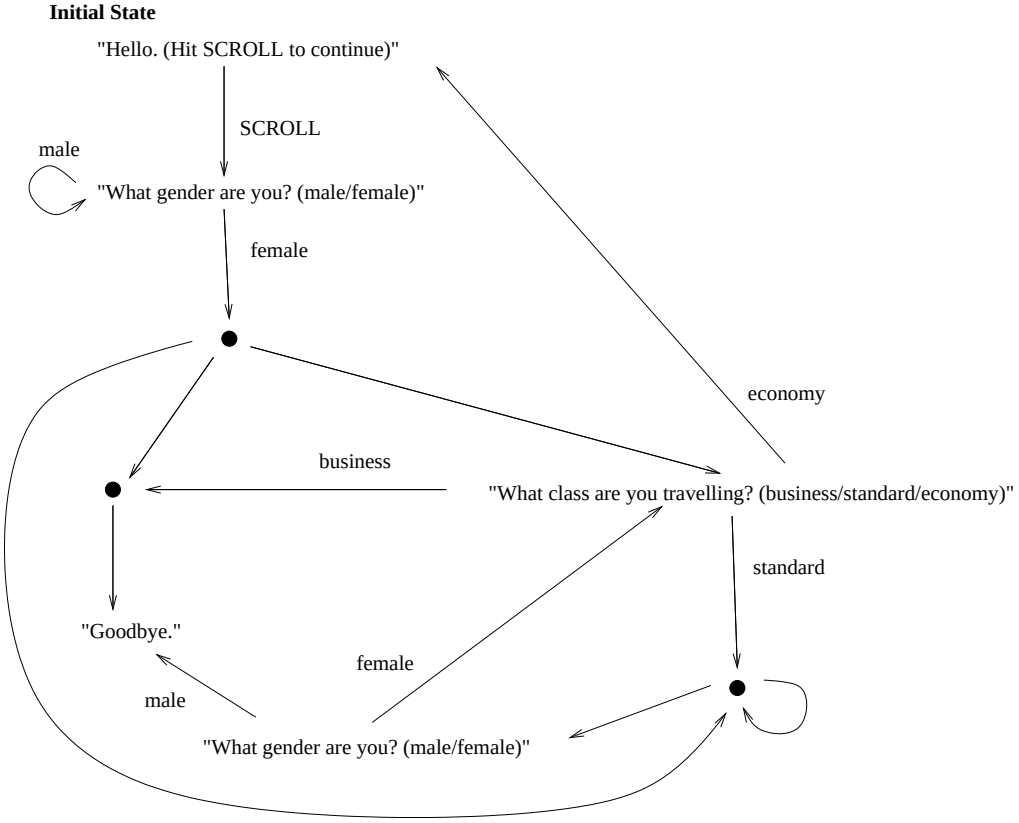


Fig. 1. Two programs illustrating interactive input

The usual adequacy proof for game semantics uses logical relations (McCusker, 1996; Pitts, 1996), but that only works for divergence-least semantics, which ours is not. Instead, we prove adequacy using a novel method. The idea is that it is easy to prove adequacy for deterministic, divergence-free terms; and every term can be converted into such a term using an "unhiding" transform, which makes every step of execution visible. That gives a highly extensional semantics, from which we can recover the desired semantics by hiding all these visible steps. We then deduce adequacy for each term from the known adequacy for its unhiding.

1.4 Diagrammatic Statement of Computational Adequacy

There is a diagrammatic description of adequacy that will be useful for our purposes. Take PCF for example. Write $\mathsf{PCF}(B)$ for the set of closed terms of type B, and $\grave{\mathsf{PCF}}(B)$ for the set of such terms that are terminal (where evaluation terminates). The operational semantics of PCF provides, for each type B, a function

$$\mathsf{PCF}(B) \xrightarrow{\Downarrow_B} T\,\grave{\mathsf{PCF}}(B)$$

where T is the *lifting monad* on $\mathbf{Set}$ that adds an extra element $\perp$.

In any particular model of PCF, the denotation of the judgement $\vdash B$ will be a T-algebra (X_B, θ_B), i.e. a pointed set. Thus each term $\vdash M : B$ denotes an element $[\![M]\!] \in X$. Computational adequacy amounts to the commutativity of the following, for each type B.

$$\begin{array}{ccc} \mathsf{PCF}(B) & \xrightarrow{\Downarrow_B} & T\,\grave{\mathsf{PCF}}(B) \\ {\scriptstyle [\![-]\!]}\downarrow & & \downarrow{\scriptstyle T[\![-]\!]} \\ X_B & \xleftarrow[\theta_B]{} & TX_B \end{array} \tag{1}$$

This says that if $M \Downarrow T$ then $[\![M]\!] = [\![T]\!]$ and if M diverges then $[\![M]\!] = \perp$.

For languages with other computational effects [4], such as nondeterminism and I/O, this notion of adequacy is still a reasonable one, although T will be not lifting but some other (inclusion-preserving) monad on $\mathbf{Set}$ appropriate to those effects.

1.5 Related Work

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.)

Adequacy of cpo-enriched semantics in the presence of algebraic effects (such as interactive input and erratic nondeterminism) is studied in (Plotkin and Power, 2001). The form of the operational semantics resembles our unhiding transform in that each operation (in particular, erratic choice) is made into an explicit action.

[4] other than control effects, for which this formulation does not make sense

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

The language of Sect. 1.1 contained an erratic choice operator $\mathtt{choose}$ of arity $\mathbb{N}$. In this section, we generalize this by having an entire family of erratic choice operators $\{\mathtt{choose}^h\}_{h\in H}$ where the arity of $\mathtt{choose}^h$ is given by a set P_h.

We thus define an *erratic signature* to be a family of sets $Y = \{P_h\}_{h\in H}$. Such a signature, together with an alphabet $\mathcal{A}$, determines a calculus $\mathcal{L}(\mathcal{A}, Y)$ with syntax

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

where c ranges over $\mathcal{A}$, and h ranges over H, and e ranges over E. The command $\mathtt{choose}^h\{M_p\}_{p\in P_h}$ means: erratically choose some $p \in P_h$, then execute M_p.

A signature in which P_h is non-empty for every $h \in H$ is said to be *lively*. According to the explanation just given, the calculus does not make computational sense if the erratic signature is not lively. Nonetheless, we will consider both lively and non-lively signatures in this paper; we justify studying the latter in Sect. 8.1.

Let Y be an erratic signature and $\mathcal{A}$ an alphabet. For each context $\Gamma = \mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$, we define a terminable [5] LTS $\mathcal{L}(\mathcal{A}, Y, \Gamma)$ with labels $\mathcal{A} + \{\tau\}$. Its states are the terms $\Gamma \vdash M$ built using Y and $\mathcal{A}$, and its terminal states are the free identifiers. The transitions are

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

For a closed term M, we say that

[5] A *terminable LTS* is a labelled transition system (LTS) in which some states are designated terminal, and there is no transition from a terminal state.

- $a_0, \ldots, a_{n-1} \in \mathcal{A}^*$ is a *finite trace* of M when $M \overset{\tau^* a_0 \tau^* \cdots \tau^* a_{n-1}}{\rightsquigarrow} N$ for some N
- $a_0, \ldots, a_{n-1} \in \mathcal{A}^*$ is a *divergence* of M when $M \overset{\tau^* a_0 \tau^* \cdots \tau^* a_n \tau^\omega}{\rightsquigarrow}$
- $a_0, a_1, \ldots \in \mathcal{A}^\omega$ is an *infinite trace* of M when $M \overset{\tau^* a_0 \tau^* a_1 \cdots}{\rightsquigarrow}$

We say that two closed terms M, M' are *infinite trace equivalent* when they have the same finite traces, divergences and infinite traces. If the erratic signature is lively (the main case of interest), then the finite traces are redundant because they are precisely the finite prefixes of the divergences and infinite traces. We defer to Sect. 8.2 the justification for including the finite traces in the non-lively case.

The finite traces, divergences and infinite traces of an *open* term $\Gamma \vdash M$ are defined the same way. We also say that $a_0, \ldots, a_{n-1}, \mathtt{x}$ is a *terminating trace* of M when $M \overset{\tau^* a_0 \tau^* \cdots \tau^* a_{n-1}}{\rightsquigarrow} \mathtt{x}$. Two terms $\Gamma \vdash M, M'$ are infinite trace equivalent when they have the same finite and terminating traces, divergences and infinite traces. As we shall see in Sect. 2.5, this is a congruence and can be modelled denotationally.

2.2 Interactive Input

For the second extension (see Sect. 1.3), we consider interactive input (Fig. 1). We want to have a family of interactive input operators $\{\mathtt{input}^o\}_{o \in O}$. Each $o \in O$ is a message that requests input from the set I_o. We thus define an *input signature* to be a family of sets $\{I_o\}_{o \in O}$. Given an input signature $Z = \{I_o\}_{o \in O}$ and an erratic signature $Y = \{P_h\}_{h \in H}$, we obtain a calculus $\mathcal{L}(Z, Y)$ with syntax

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

where h ranges over H, and o ranges over O. (We are not including `print` explicitly, as we explain presently.)

The command $\mathtt{input}^o\{M_i\}_{i \in I_o}$ has the following meaning:

(1) print o
(2) wait until the user inputs some $i \in I_o$
(3) execute M_i

If the user never supplies input, the program will wait forever.

Two cases of input operator are of special interest: unary and nullary.

- Where I_o is singleton, the command $\mathtt{input}^o\{M\}$ prints o, waits for a specified input (the user hitting a SCROLL button, let us say), and then continues to execute M. This is *slightly* different from $\mathtt{print}\ o.\ M$, which executes M immediately after printing o. However, for the purposes of this paper, we regard them

as the same thing; therefore no `print` primitive is required in the calculus.
- Where I_o is empty, the command $\texttt{input}^o\{\}$ simply prints the message o, and nothing further can happen. In effect, this command throws an unrecoverable error, and o is the error message.

Remark 1 Interactive input using input signature $Z = \{I_o\}_{o \in O}$ is an example of a *computational effect* (Moggi, 1991; Plotkin and Power, 2002), represented as a monad on $\mathbf{Set}$, viz. the free monad on the endofunctor R_Z on $\mathbf{Set}$ defined by $X \mapsto \sum_{o \in O} X^{I_o}$. Explicitly, this monad maps a set V to $\mu Y.(V + R_Z Y)$.

Three monads appearing in (Moggi, 1991) are special cases of this, following (Plotkin and Power, 2002).

- The interactive input monad $V \mapsto \mu Y.(V + Y^I)$ arises from the input signature with one operator of arity I.
- The interactive output monad $V \mapsto \mathcal{A}^* \times V$ arises from the signature with $\mathcal{A}$ unary operators.
- The exceptions monad $V \mapsto V + E$ arises from the signature with E nullary operators.

□

2.3 Operational Semantics of Interactive Input

In Sect. 2.1, we gave the operational semantics of a printing calculus as a terminable LTS. But for a calculus with interactive input, this is not quite suitable:

- If we allow both outputs and inputs to be actions, we need additional alternation and receptivity-to-input conditions.
- 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 need a transition system of the kind depicted in Fig. 1, though without an initial state.

Definition 1 (BLTS) Let $Z = \{I_o\}_{o \in O}$ be an I/O signature.

(1) A *bi-labelled transition system* (BLTS) $\mathcal{M}$ over Z consists of
- a set (which we also call $\mathcal{M}$) of *states*, each of which is classified as either *o-interactive* for some $o \in O$, or *silent*
- for each o-interactive state d, and each input $i \in I_o$, a state $d : i \in \mathcal{M}$.
- for each silent state d, a set of *successors* $\mathsf{succ}(d) \subseteq \mathcal{M}$

We write $d \downarrow o$ when d is an o-interactive state. We write $d \rightsquigarrow d'$ when d is silent and $d' \in \mathsf{succ}(d)$.

(2) A *terminable BLTS* $\mathcal{M}$ is the same, except that there is a third kind of state: *terminal*. We write $\grave{\mathcal{M}}$ for the set of terminal states.
(3) A BLTS or terminable BLTS is *lively* when each silent state has at least one successor, and *deterministic* when each silent state has precisely one successor.

□

Remark 2 Defining R_Z as in Remark 1, we can, more abstractly, define a BLTS over Z to be a coalgebra for the endofunctor $X \mapsto \mathcal{P}X + R_Z X$ on $\mathbf{Set}$. □

Let Z be an input signature and Y an erratic signature. For each context $\Gamma = \mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$, we define a terminable BLTS $\mathcal{L}(Z, Y, \Gamma)$ over Z as follows. The states are the terms $\Gamma \vdash M$ in the calculus $\mathcal{L}(Z, Y)$, with transitions given in Fig. 2. In particular, the terminal states are the free identifiers.

Interactive commands

$$\mathtt{input}^o\{M_i\}_{i \in I_o} \quad \downarrow o$$

Interactive transitions

$$\mathtt{input}^o\{M_i\}_{i \in I_o} \quad : \hat{\imath} \;=\; M_{\hat{\imath}} \qquad (\hat{\imath} \in I_o)$$

Silent commands

$$\mathtt{rec}\ \mathtt{x}.\ M$$

$$\mathtt{choose}^h\{M_p\}_{p \in P_h}$$

Silent transitions

$$\mathtt{rec}\ \mathtt{x}.\ M \quad \rightsquigarrow \quad M[\mathtt{rec}\ \mathtt{x}.\ M/\mathtt{x}]$$

$$\mathtt{choose}^h\{M_p\}_{p \in P_h} \quad \rightsquigarrow \quad M_{\hat{p}} \qquad (\hat{p} \in P_h)$$

Terminal commands

$$\mathtt{x} \qquad (\mathtt{x} \in \Gamma)$$

Fig. 2. Operational semantics of $\mathcal{L}(Z, Y)$ as terminable BLTS $\mathcal{L}(Z, Y, \Gamma)$

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 $\mathrm{y} \in \Gamma$, we have $M = \mathrm{y}$ iff $M[N/\mathrm{x}] = \mathrm{y}$.

□

2.4 Strategies in a BLTS

As in Sect. 2.1, we can define finite traces, divergences and infinite traces. Fix an input signature $Z = \{I_o\}_{o \in O}$.

Definition 2 Let $Z = \{I_o\}_{o \in O}$ be an input signature. A *play* over Z is a finite or infinite sequence $o_0 i_0 o_1 i_1 \ldots$ where $o_r \in O$ and $i_r \in I_{o_r}$ for each r. It *awaits Proponent* if of even length, and *awaits o-input* if of odd length ending in o. □

Definition 3 Let d be a state within a BLTS $\mathcal{M}$ over Z.

(1) An input-awaiting play $o_0 i_0 \ldots o_{n-1} i_{n-1} o_n$ is a *finite trace* of d when there is a sequence of states

$$\begin{aligned} d &\rightsquigarrow^* e_0 \downarrow o_0 \\ e_0 : i_0 &\rightsquigarrow^* e_1 \downarrow o_1 \\ &\vdots \\ e_{n-1} : i_{n-1} &\rightsquigarrow^* e_n \downarrow o_n \end{aligned}$$

(2) A Proponent-awaiting play $o_0 i_0 \ldots o_{n-1} i_{n-1}$ is a *divergence* of d when there is a sequence of states

$$\begin{aligned} d &\rightsquigarrow^* e_0 \downarrow o_0 \\ e_0 : i_0 &\rightsquigarrow^* e_1 \downarrow o_1 \\ &\vdots \\ e_{n-1} : i_{n-1} &\rightsquigarrow^\omega \end{aligned}$$

(3) An infinite play $o_0 i_0, o_1, i_1, \ldots$ is an *infinite trace* of d when there is a sequence of states

$$\begin{aligned} d &\rightsquigarrow^* e_0 \downarrow o_0 \\ e_0 : i_0 &\rightsquigarrow^* e_1 \downarrow o_1 \\ e_1 : i_1 &\rightsquigarrow^* e_2 \downarrow o_2 \\ &\vdots \end{aligned}$$

□

Of course, any finite prefix of a finite trace, divergence or infinite trace of s is a finite trace. So we make the following definition.

Definition 4 (1) A *strategy* over Z consists of

- a set A of input-awaiting plays
- a set B of Proponent-awaiting plays
- a set C of infinite plays

such that every input-awaiting prefix of a play in $A \cup B \cup C$ is in A.

(2) Let d be a state in a BLTS $\mathcal{M}$ over Z. The *operational meaning* of d, written $[d]$, is the strategy over Z given by the finite traces, divergences and infinite traces of d. Two states d and d' are *infinite trace equivalent* when $[d] = [d']$.

□

In the case of a terminable BLTS, there is a fourth kind of behaviour we need to consider.

Definition 5 (1) Let V be a set. A *V-terminating play* over Z is a sequence $o_0 i_0 \ldots o_{n-1} i_{n-1} v$ where $o_r \in O$ and $i_r \in I_{o_r}$ for each r, and $v \in V$.

(2) Let d be a state within a terminable BLTS $\mathcal{M}$ over Z. (Recall that $\grave{\mathcal{M}}$ is the set of terminal states of $\mathcal{M}$.) A $\grave{\mathcal{M}}$-terminating play $o_0 i_0 \ldots o_{n-1} i_{n-1} v$ is a *terminating trace* of d when there is a sequence of states

$$\begin{aligned} d &\rightsquigarrow^* e_0 \downarrow o_0 \\ e_0 : i_0 &\rightsquigarrow^* e_1 \downarrow o_1 \\ &\vdots \\ e_{n-1} : i_{n-1} &\rightsquigarrow^* v \end{aligned}$$

The input-awaiting traces, divergences and infinite traces of s are defined as for a state of a BLTS. A *finite trace* is either an input-awaiting trace or a terminating trace.

(3) Let V be a set. A *V-terminable strategy* over Z consists of

- a set $A = A_{\mathsf{input}} \cup A_{\mathsf{termin}}$ of input-awaiting and V-terminating plays
- a set B of Proponent-awaiting plays
- a set C of infinite plays

such that any input-awaiting prefix of $A \cup B \cup C$ is in A_{input}.

(4) Let $\mathcal{M}$ be a terminable BLTS over Z. For any state $d \in \mathcal{M}$, the *operational meaning* of d, written $[d]$, is the $\grave{\mathcal{M}}$-terminable strategy over Z given by the finite traces, divergences and infinite traces of d.

□

Definition 6 Let V be a set. We build V-terminable strategies over Z using the following operations.

(1) For $v \in V$, we define ηv to be the strategy

$$(\{v\}, \{\}, \{\})$$

(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

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

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

$$(\{o\} \cup \{oil | i \in I_o, l \in A_i\}, \{oil | i \in I_o, l \in B_i\}, \{oil | i \in I_o, l \in C_i\})$$

□

Proposition 1 Let d be a state in a terminable BLTS $\mathcal{M}$ over Z. Let V be the set of terminal states.

- 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 $[d] = \eta d$.

□

2.5 Denotational Semantics

The key result of this section is that, on the terminable BLTS $\mathcal{L}(Z, Y, \Gamma)$, we can characterize $[-]$ in a compositional way.

Proposition 2 In the language $\mathcal{L}(Y, Z)$, we have the following.

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

$$[\mathtt{rec\ x}.\ M]_{\mathcal{L}(Z,Y,\Gamma)} = \mu[M]_{\mathcal{L}(Z,Y,\Gamma,\mathtt{x})}$$

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

$$\begin{aligned}
&(\{l_0 \cdots l_{n-1} l | l_0\mathtt{x} \in A_{\mathsf{termin}}, \ldots, l_{n-1}\mathtt{x} \in A_{\mathsf{termin}}, l \in A_{\mathsf{input}}\} \\
&\quad \cup \{l_0 \cdots l_{n-1} l\mathtt{y} | l_0\mathtt{x} \in A_{\mathsf{termin}}, \ldots, l_{n-1}\mathtt{x} \in A_{\mathsf{termin}}, l\mathtt{y} \in A_{\mathsf{termin}}, \mathtt{y} \neq \mathtt{x}\}, \\
&\{l_0 \cdots l_{n-1} l | l_0\mathtt{x} \in A_{\mathsf{termin}}, \ldots, l_{n-1}\mathtt{x} \in A_{\mathsf{termin}}, l \in B\} \\
&\quad \cup \{l_0 \cdots l_{n-1} | l_0\mathtt{x} \in A_{\mathsf{termin}}, \ldots, l_{n-1}\mathtt{x} \in A_{\mathsf{termin}}, \epsilon\mathtt{x} \in A_{\mathsf{termin}}\}, \\
&\{l_0 \cdots l_{n-1} l | l_0\mathtt{x} \in A_{\mathsf{termin}}, \ldots, l_{n-1}\mathtt{x} \in A_{\mathsf{termin}}, l \in C\} \\
&\quad \cup \{l_0 l_1 \cdots | l_0\mathtt{x} \in A_{\mathsf{termin}}, l_1\mathtt{x} \in A_{\mathsf{termin}}, \ldots \text{ and } \forall i \in \mathbb{N}.\ l_i \neq \epsilon\})
\end{aligned}$$

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

$$[M[N/\mathtt{x}]]_{\mathcal{L}(Z,Y,\Gamma)} = [M]_{\mathcal{L}(Z,Y,\Gamma,\mathtt{x})} * [N]_{\mathcal{L}(Z,Y,\Gamma)}$$

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

$$\begin{aligned}(A_{\mathsf{input}} \cup \{l\mathtt{y} \mid l\mathtt{y} \in A_{\mathsf{termin}}, \mathtt{y} \neq \mathtt{x}\} \cup \{ll' | l\mathtt{x} \in A_{\mathsf{termin}}, l' \in A'\},\\ B \cup \{ll' | l\mathtt{x} \in A_{\mathsf{termin}}, l' \in B'\},\\ C \cup \{ll' | l\mathtt{x} \in A_{\mathsf{termin}}, l' \in C'\})\end{aligned}$$

□

We thus define a denotational model $[\![\mathtt{rec}\ \mathtt{x}.\ M]\!] = \mu[\![M]\!]$ etc., and Prop. 2(1)–(4) shows computational adequacy i.e. $[\![M]\!] = [M]$.

3 Monads and Algebraic Operations

Let $Z = \{I_o\}_{o \in O}$ be an input signature.

3.1 The Monad of Nondeterministic Strategies

For any set V, we write $T_Z(V)$ for the set of V-terminable strategies over Z. This gives us a monad on $\mathbf{Set}$—it is the monad representing the combination of interactive input over Z, nondeterminism and divergence, under infinite trace equivalence. The unit at V is given by Def. 6(1). The multiplication at V maps $(A, B, C) \in T_Z T_Z V$ to

$$\begin{aligned}(A_{\mathsf{input}} \cup \{ll' \mid l(A', B', C') \in A_{\mathsf{termin}}, l' \in A'\},\\ B \cup \{ll' \mid l(A', B', C') \in A_{\mathsf{termin}}, l' \in B'\},\\ C \cup \{ll' \mid l(A', B', C') \in A_{\mathsf{termin}}, l' \in C'\})\end{aligned} \tag{2}$$

The monad laws are easily verified.

For any terminable BLTS $\mathcal{M}$, Def. 5(4) gives us a function $\mathcal{M} \xrightarrow{[-]} T_Z\grave{\mathcal{M}}$. This takes the place of $\Downarrow$ in Sect. 1.4.

3.2 Algebraic Operations

We can define $\bigcup$ and input^o in a general setting.

Definition 7 Let $\underline{X} = (X, \theta)$ be a T_Z-algebra.

(1) Given a family $\{x_i\}_{i \in I}$ of elements of X, we define

$$\bigcup_{i \in I} x_i \stackrel{\text{def}}{=} \theta(\{x_i \mid i \in I\}, \{\}, \{\})$$

(2) Given $o \in O$ and a family $\{x_i\}_{i \in I_o}$ of elements of X, we define

$$\mathsf{input}^o\{x_i\}_{i \in I_o} \stackrel{\text{def}}{=} \theta(\{o\} \cup \{oix_i \mid i \in I_o\}, \{\}, \{\})$$

□

If we apply Def. 7 to the free algebra on V, we recover the constructions given in Def. 6(2)–(3).

We recall [6] the following concept from (Plotkin and Power, 2003).

Definition 8 Let T be a monad on $\mathbf{Set}$ and let I be a set. An I-ary *algebraic operation* α for T provides, for each T-algebra $\underline{X} = (X, \theta)$, a function

$$X^I \xrightarrow{\alpha_X} X \qquad \text{natural in } \underline{X} \in \mathbf{Set}^T.$$

□

It is easy to see that Def. 7 gives us algebraic operations for T_Z.

- For each set I, the operation $\bigcup_{i \in I}$ is an I-ary algebraic operation.
- For each $o \in O$, the operation $\mathsf{input}^o_{i \in I_o}$ is an I_o-ary algebraic operation.

4 Jump-With-Argument With Type Recursion

4.1 The Language

We now want to move to a language with higher-order types. One possibility to simply add higher-order types to the language $\mathcal{L}(Z, Y)$, as done in (Levy, 2006b). However, giving game semantics directly for a call-by-name language is quite complicated. To make the game semantics as easy as possible, we use a continuation passing calculus "Jump-With-Argument" (JWA).

[6] Although the initial formulation in (Plotkin and Power, 2003) covers free T-algebras only, it is shown that an algebraic operation over free algebras extends uniquely to one over all algebras. So we take the latter as our definition, cf. (Møgelberg and Simpson, 2007).

We can then use a "stack passing" transform (Levy, 2004a) to translate call-by-push-value, a calculus that subsumes call-by-name and call-by-value (Levy, 2006a), into JWA. On the call-by-value fragment, this is the traditional CPS transform, while on the call-by-name fragment, it is the transform given in (Streicher and Reus, 1998). A categorical description of how, from a model of JWA, we can construct a model of call-by-push-value is given in (Levy, 2005)

The types of JWA with type recursion are given by

$$A ::= \quad \neg A \mid \textstyle\sum_{i \in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mu \mathtt{X}.A$$

where I ranges over countable sets. (We can also consider a finitary version, where I ranges over finite sets.) The type $\neg A$ is the type of functions that take an argument of type A and do not return.

More formally, if Φ is a type context (list of type identifiers), we write $\Phi \vdash^{\mathsf{type}} A$ to mean that A is a type whose free identifiers are included in Φ. This is defined inductively in the usual way.

JWA has two kinds of term: *values* and *nonreturning commands*, indicated by the judgements $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma \vdash^{\mathsf{n}} M$ respectively. The types in Γ and the type A must all be closed.

For a given input signature $Z = \{I_o\}_{o \in O}$ and an erratic signature $Y = \{P_h\}_{h \in H}$, we define $\mathsf{JWA}(Z, Y)$, i.e. JWA extended with type recursion, interactive input from Z and erratic choice from Y. The syntax is given in Fig. 3. We write `pm` as an abbreviation for "pattern-match", and write `let` to make a binding. We omit typing rules, etc., for 1, since 1 is analogous to $\times$.

The operational semantics is given in the same style as in Fig. 2: for each context Γ we define a terminable BLTS $\mathsf{JWA}(Z, Y, \Gamma)$ over Z. The states are the commands $\Gamma \vdash^{\mathsf{n}} M$ in $\mathsf{JWA}(Z, Y)$. The transitions are shown in Fig. 4.

To translate the language $\mathcal{L}(Z, Y)$ into $\mathsf{JWA}(Z, Y)$, a command $\mathtt{x}_0, \ldots, \mathtt{x}_{n-1} \vdash M$ is translated into a command $\mathtt{x}_0 : \neg 1, \ldots \mathtt{x}_{n-1} : \neg 1 \vdash^{\mathsf{n}} \overline{M}$. In particular, a free identifier $\mathtt{x}$ is translated as $\mathtt{x}\langle\rangle$. Recursion can be encoded in terms of type recursion in the usual way; we omit details.

4.2 Categorical semantics of JWA

It it usual, and convenient, to use categorical structure to organize game models, rather than interpreting syntax directly. In this section, we recall from (Levy, 2005) the relevant categorical structure for JWA.

Firstly, if $\mathcal{C}$ is a category, a *left $\mathcal{C}$-module* is a functor $\mathcal{N} : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}$. An element

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{x} : A}\ (\mathtt{x} : A) \in \Gamma \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{n}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M}$$

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

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

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{v}} \lambda \mathtt{x}.M : \neg A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \neg A \quad \Gamma \vdash^{\mathsf{v}} W : A}{\Gamma \vdash^{\mathsf{n}} VW}$$

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

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

Fig. 3. Syntax of the calculus $\mathsf{JWA}(Z, Y)$

$g \in \mathcal{N}(R)$ is called an $\mathcal{N}$-*morphism from* R (though it is not a morphism *to* anything), and written $R \xrightarrow{g}$. Given a $\mathcal{C}$-morphism $R \xrightarrow{f} S$ and an $\mathcal{N}$-morphism $S \xrightarrow{g}$, we define the composite $R \xrightarrow{f} S \xrightarrow{g}$ to be $\mathcal{N}_f g$.

A cartesian category $\mathcal{C}$ together with a left $\mathcal{C}$-module $\mathcal{N}$ is called a *JWA judgement model*, because it can be used to interpret the $1, \times$ fragment of JWA, in the following manner.

- A type denotes a $\mathcal{C}$-object
- A context $\Gamma = A_0, \ldots, A_{n-1}$ denotes the $\mathcal{C}$-object $[\![\Gamma]\!] = [\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$
- A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$
- A command $\Gamma \vdash^{\mathsf{n}} M$ denotes a $\mathcal{N}$-morphism from $[\![\Gamma]\!]$.

Given a JWA judgement model $(\mathcal{G}, \mathcal{S})$, the *families construction* (Abramsky and McCusker, 1998) gives us another one which we call $(\mathsf{fam}_\omega \mathcal{G}, \mathsf{fam}_\omega \mathcal{S})$. A $\mathsf{fam}_\omega \mathcal{G}$-object is a countable family of $\mathcal{G}$-objects. We define

Interactive commands

$$\mathtt{input}^o\{M_i\}_{i\in I_o} \qquad \downarrow o$$

Interactive transitions

$$\mathtt{input}^o\{M_i\}_{i\in I_o} \qquad : \hat{\imath} \;=\; M_{\hat{\imath}} \qquad (\hat{\imath} \in I_o)$$

Silent commands

$$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$$

$$\mathtt{pm}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i, \mathtt{x}_i\rangle.M_i\}_{i\in I} \qquad (\hat{\imath} \in I)$$

$$\mathtt{pm}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M$$

$$(\lambda \mathtt{x}.M)V$$

$$\mathtt{pm}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M$$

$$\mathtt{choose}^h\{M_p\}_{p\in P_h}$$

Silent transitions

$$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.M \rightsquigarrow M[V/\mathtt{x}]$$

$$\mathtt{pm}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i\in I} \rightsquigarrow M_{\hat{\imath}}[V/\mathtt{x}] \qquad (\hat{\imath} \in I)$$

$$\mathtt{pm}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M \rightsquigarrow M[V/\mathtt{x}, V'/\mathtt{y}]$$

$$(\lambda \mathtt{x}.M)V \rightsquigarrow M[V/\mathtt{x}]$$

$$\mathtt{pm}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M \rightsquigarrow M[V/\mathtt{x}]$$

$$\mathtt{choose}^h\{M_p\}_{p\in P_h} \rightsquigarrow M_{\hat{p}} \qquad (\hat{p} \in P_h)$$

Terminal commands

$$\mathtt{z}V \qquad \text{where } (\mathtt{z} : \neg A) \in \Gamma$$

$$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\ M_i\}_{i\in I} \qquad \text{where } (\mathtt{z} : \textstyle\sum_{i\in I} A_i) \in \Gamma$$

$$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M \qquad \text{where } (\mathtt{z} : A \times A') \in \Gamma$$

$$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M \qquad \text{where } (\mathtt{z} : \mu \mathtt{X}.A) \in \Gamma$$

Fig. 4. Operational semantics of $\mathsf{JWA}(Z, Y)$ commands in context Γ

$$(\mathsf{fam}_\omega \mathcal{G})(\{R_i\}_{i\in I}, \{S_j\}_{j\in J}) \stackrel{\text{def}}{=} \prod_{i\in I}\sum_{j\in J} \mathcal{G}(R_i, S_j) \tag{3}$$

$$(\mathsf{fam}_\omega \mathcal{S})(\{R_i\}_{i\in I}) \stackrel{\text{def}}{=} \prod_{i\in I} \mathcal{S}(R_i) \tag{4}$$

$$\{R_i\}_{i\in I} \times \{S_j\}_{j\in J} \stackrel{\text{def}}{=} \{R_i \times S_j\}_{\langle i,j\rangle \in I\times J} \tag{5}$$

and define composition and identities in the obvious way.

The structure $(\mathsf{fam}_\omega\mathcal{G}, \mathsf{fam}_\omega\mathcal{S})$ always provides a model of the $\times, 1, \sum$ fragment of JWA, using

$$\textstyle\sum_{i\in I}\{R_{ij}\}_{j\in J_i} = \{R_{ij}\}_{\langle i,j\rangle\in \sum_{i\in I} J_i}$$

But to be able to model $\neg$, we need additional structure on $(\mathcal{G}, \mathcal{S})$, as we explain.

Definition 9 A *JWA pre-families structure* consists of

- a JWA judgement model $(\mathcal{G}, \mathcal{S})$
- for each countable family of $\mathcal{G}$-objects $\{R_i\}_{i\in I}$, a representing object for the functor

$$\prod_{i\in I} \mathcal{S}(- \times R_i) : \mathcal{G}^{\mathsf{op}} \to \mathbf{Set}$$

  i.e. an object $\neg_{i\in I} R_i$ together with an isomorphism

$$\prod_{i\in I} \mathcal{S}(X \times R_i) \cong \mathcal{G}(X, \neg_{i\in I} R_i) \qquad \text{natural in } X \in \mathcal{G}^{\mathsf{op}}.$$

□

If $(\mathcal{G}, \mathcal{S})$ is a JWA pre-families structure, then $(\mathsf{fam}_\omega\mathcal{G}, \mathsf{fam}_\omega\mathcal{S})$ is a model of JWA. A $\neg$ type denotes a singleton family:

$$\neg(\{R_i\}_{i\in I}) = \{\neg_{i\in I} R_i\}$$

4.3 Enriched Models of JWA

In order to model JWA extended with computational effects, we need additional structure.

Definition 10 Let T be a monad on $\mathbf{Set}$.

(1) A *T-enriched JWA judgement model* is a cartesian category $\mathcal{C}$ together with a functor $\underline{\mathcal{N}} : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}^T$, where $\mathbf{Set}^T$ is the category of T-algebras and homomorphisms. Equivalently, it is a JWA judgement model $(\mathcal{C}, \mathcal{N})$ together with a natural transformation $T\mathcal{N} \xrightarrow{\beta} \mathcal{N}$ such that $(\mathcal{N}A, \beta A)$ is a T-algebra for every $A \in \mathsf{ob}\,\mathcal{C}$.

(2) If $(\mathcal{G}, \underline{\mathcal{S}})$ is a T-enriched JWA judgement model, then we define another T-enriched JWA judgement model $(\mathsf{fam}_\omega\mathcal{G}, \mathsf{fam}_\omega\underline{\mathcal{S}})$ by setting

$$(\mathsf{fam}_\omega\underline{\mathcal{S}})(\{R_i\}_{i\in I}) \stackrel{\text{def}}{=} \prod_{i\in I} \underline{\mathcal{S}}(R_i)$$

as in (4), but here we are taking the product of T-algebras.

□

Let $(\mathcal{C}, \mathcal{N}, \beta)$ be a T-enriched judgement model. Any I-ary algebraic operation α for T induces a map

$$(\mathcal{N}A)^I \xrightarrow{\tilde{\alpha}A} \mathcal{N}A \qquad \text{natural in } A \in \mathcal{C}$$

where $\tilde{\alpha}A \stackrel{\text{def}}{=} \alpha(\mathcal{N}A, \beta A)$. Thus in the case of the monad T_Z, we obtain $\bigcup$ and input^o constructions on the $\mathcal{N}$ homsets. We can use these to interpret JWA with input signature Z and any erratic signature.

5 Pointer Games

5.1 Arenas

In this section we describe the pointer game semantics for JWA, adapting the deterministic semantics given in (Levy, 2005). We assume the reader is familiar from (Abramsky et al., 1998; Hyland and Ong, 2000; Nickau, 1996) with this style of semantics, so we do not motivate it here; see (Lassen and Levy, 2007) for an operational theory that is closely connected.

Definition 11 • An *arena* is a countable set R equipped with a relation $\vdash \subseteq (\{*\} + R) \times R$ that depicts a forest, i.e. for each $r \in R$ there is a unique finite sequence

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

The *roots* of R are the elements $\mathsf{rt}\ R \stackrel{\text{def}}{=} \{r \in R \mid * \vdash r\}$, and the *children* of $s \in R$ are the elements $\{r \in R \mid s \vdash r\}$.

- We write $R \uplus S$ for the disjoint union of R and S, and $\emptyset$ for the empty arena.
- For a countable family of arenas $\{R_i\}_{i \in I}$, we write $\mathsf{pt}_{i \in I} R_i$ for the arena with I roots and a copy of R_i placed below the ith root.
- If $r \in R$, we write $R{\restriction_r}$ for the arena of elements *strictly* descended from r.

□

Although it is not usually made explicit in the game literature, the following category is important, as it is used for coherence isomorphisms.

Definition 12 A *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 (and indeed countable, though it is only finite that we use) coproducts given by disjoint union. □

5.2 Pointer Game: informal definition

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

- Play alternates between Proponent and Opponent, with Proponent moving first.
- In each move, an element of R is played.
- Proponent 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 Proponent-move m and stating a child of the element played in m.

We write $\mathcal{S}R$ for the set of nondeterministic strategies for this game (we define this more formally presently). We then set up a JWA pre-families structure $(\mathcal{G}, \mathcal{S})$. The objects of $\mathcal{G}$ are arenas, with finite products given by $\uplus$ and $\neg$ structure given by $\mathsf{pt}_{i \in I}$. The homsets are given by

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

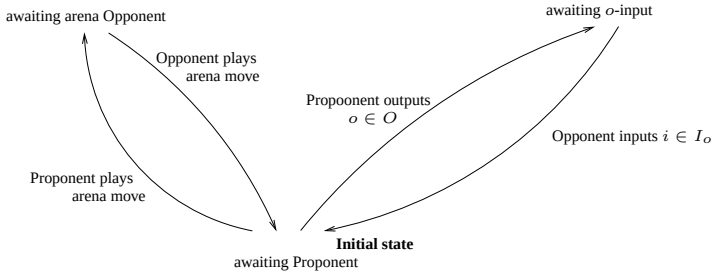
for all arenas R and S. And we will then define identity maps, both kinds of composition, etc., in the usual way.

Remark 3 $\mathcal{G}$ is the category defined in (Hyland and Ong, 2000), minus the constraints of innocence, visibility, bracketing and determinism. The question/answer labelling is omitted, as it is redundant in the absence of the bracketing condition. □

The structure $(\mathsf{fam}_\omega \mathcal{G}, \mathsf{fam}_\omega \mathcal{S})$ will be used to model JWA with nondeterminism but without I/O. In order to model JWA with an input signature $Z = \{I_o\}_{o \in O}$, we modify the pointer game on R:

- Proponent has a third option for playing a move: to output some $o \in O$
- Opponent then responds with some $i \in I_o$
- play continues as usual.

This is depicted as follows:



Taking nondeterministic strategies for this variant game, we obtain a JWA pre-families structure $(\mathcal{G}_Z, \mathcal{N}_Z)$. We shall see that it is T_Z-enriched.

Fix an input signature $Z = \{I_o\}_{o \in O}$. We are going to define strategies wrt this signature.

Definition 13 A *justified sequence* s in an arena R over Z consists of

- a finite or infinite sequence $s_0 s_1 s_2 \ldots$ where each s_m is either
  - an element of R (we say that m is an *arena move*)
  - an element $o \in O$ (we say that m is an o*-output move*)
  - an element of I_o, for some $o \in O$ (we say that m is an o*-input move*)

  where s_m is an o-input move iff it follows an o-output move
- for each arena move m such that s_m is not a root, a pointer to a move ptr_m such that $\mathsf{ptr}_m < m$ and $s_{\mathsf{ptr}_m} \vdash s_m$.

□

Pictorially, we describe an arena move m as $\mathsf{ptr}_m \curvearrowleft s_m$, where $\mathsf{ptr}_m \stackrel{\text{def}}{=} *$ in the case that m is a root move.

Definition 14 A *play* on arena R over Z is a justified sequence such that, for every move m,

- if m is even (e.g. 0) then it is either an output move, an arena move playing a root, or an arena move pointing 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.

A finite play *awaits Proponent* or *awaits Opponent* according as its length is even or odd. In the latter case, it *awaits arena-Opponent* or *awaits* o*-input* according as its last move is an arena move or an o-output move. □

Definition 15 A *strategy* on an arena R over Z consists of

- a set A of Opponent-awaiting plays (the *finite traces*)
- a set B of divergences (the *divergences*)
- a set C of infinite plays (the *infinite traces*)

such that if s is in A, B or C, then every Opponent-awaiting prefix is in A. We write $\mathcal{S}_Z R$ for the set of strategies on R over Z. □

It is clear that $\mathcal{S}_Z R$ is functorial in $R \in \mathbf{TokRen}$.

As stated above, we define $\mathcal{G}_Z(R, S) \stackrel{\text{def}}{=} \prod_{b \in \mathsf{rt}\, S} \mathcal{S}_Z(R \uplus S\restriction_b)$ for all arenas R and S.

5.4 Copycat and Composition

To define the identity morphism on R, we require for each $b \in \mathsf{rt}\ R$ a strategy $\mathsf{id}_{R,b}$ on $R \uplus R\restriction_b$. This is a "copycat" strategy: the (deterministic) strategy with no divergences, whose finite/infinite traces are all the plays in which Proponent initially plays $* \curvearrowleft \mathsf{inl}\ b$, and responds to

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

For the rest of the categorical structure, we first define an operation

$$\mathcal{G}_Z(R,S) \times \mathcal{S}_Z(S \uplus T) \xrightarrow{\divideontimes_{R,S,T}} \mathcal{S}_Z(R \uplus T)$$

for arenas R, S, T. We then define composition in terms of this:

$$R \xrightarrow{f} S \xrightarrow{g} T = \lambda c_{\in \mathsf{rt}\ T}.\ f \divideontimes_{R,S,T\restriction_c} g_c \tag{6}$$

$$R \xrightarrow{f} S \xrightarrow{g} \ = f \divideontimes_{R,S,\emptyset} g \tag{7}$$

Finally, we show that $\divideontimes$ can be recovered from the categorical structure:

Proposition 3 If $R \xrightarrow{\sigma} S$ and $R \uplus T \xrightarrow{\tau}$, then $\sigma \divideontimes_{R,S,T} \tau = (\sigma \times T); \tau$ □

c Intuitively, the strategy $\sigma \divideontimes_{R,S,T} \tau$ should execute 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 16 Let s be a justified sequence on $R \uplus S \uplus T$ wrt Z. The *inner thread initiators* of s are

$$\mathsf{inners(s)} \stackrel{\text{def}}{=} \{*\} \cup \{\text{root moves in } S\}$$

For $q \in \mathsf{inners(s)}$, the *arena of* q is $S \uplus T$ if $q = *$, and $R \uplus S\restriction_b$ if q plays $b \in \mathsf{rt}\ S$. □

Definition 17 Let s be a justified sequence on $R \uplus S \uplus T$ wrt Z, equipped with

- for each root move in R, a pointer to an earlier root move in S
- for each output move, a pointer to an inner thread initiator.

(These additional pointers are called *thread pointers*.)

(1) The *outer thread* of s is the justified sequence on $R \uplus T$ consisting of all arena moves in R and T and all I/O moves.

(2) The *inner thread initiated by* q, where $q \in \mathsf{inners}(\mathsf{s})$, is the justified sequence on the arena of q consisting of
- if $q = *$, all arena moves in S and T
- if q plays $b \in \mathsf{rt}\ S$, all the arena moves in R descended from a root move that thread-points to q, and all the arena moves in S strictly descended from q

together with the output moves that thread-point to q and the input moves that follow them.

(3) We say that s is an *interaction sequence* when all the threads (outer and inner) are plays.

□

Remark 4 The thread pointers of an interaction sequence are actually redundant. But it is more difficult to define interaction sequence without them. □

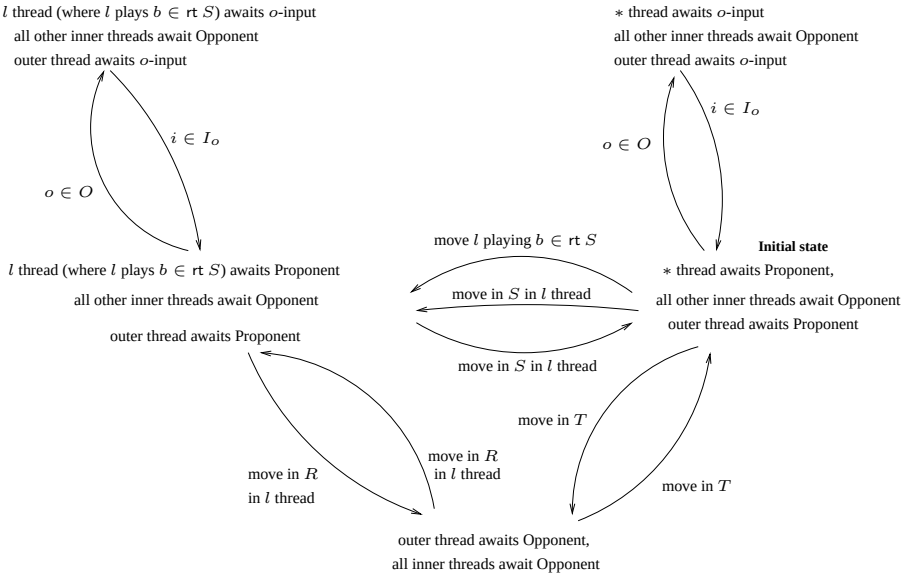


Fig. 5. State diagram followed by an interaction sequence on arenas R, S, T over Z

An interaction sequence follows the state diagram shown in Fig. 5, giving us the following result.

Proposition 4 Let s be an interaction sequence on R, S, T wrt Z. Then precisely one of the following is true.

(1) s is finite. The outer thread and every inner thread await arena Opponent.
(2) s is finite. For some $o \in O$ and $l \in \mathsf{inners}(\mathsf{s})$, the l-inner thread awaits o-input, as does the outer thread. All other inner theads await arena-Opponent.
(3) s is finite. For some $l \in \mathsf{inners}(\mathsf{m})$, the l-inner thread awaits Proponent, as does the outer thread. All other inner theads await arena-Opponent.
(4) s is infinite. Each inner thread awaits arena-Opponent or is infinite. The outer thread awaits Proponent.

(5) s is infinite. Each inner thread awaits arena-Opponent or is infinite. The outer thread is infinite.

□

We say that an interaction sequence s

- *awaits outer Opponent* in cases (1)–(2)
- *awaits l-inner Proponent* in case (3)
- is *outer-starved* in case (4)
- is *outer-infinite* in case (5).

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

Definition 18 Let R, S, T be arenas, and Z an input signature. Let $\sigma \in \mathcal{G}_Z(R, S)$ and $\tau \in \mathcal{S}_Z(S \uplus T)$. For any interaction sequence s and $q \in \mathsf{inners(s)}$, we thus have a strategy $q(\sigma, \tau)$ on the arena of q, viz. τ if $q = *$, and σ_b if q plays $b \in \mathsf{rt}\ S$.

We define $\sigma \divideontimes_{R,S,T} \tau$ to be the following strategy:

finite traces the outer thread of every outer-Opponent-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-inner-Proponent-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 whose q-inner-thread is a finite trace or infinite trace of $q(\sigma, \tau)$ for every $q \in \mathsf{inners(s)}$.

□

Proposition 5 Definition (6) satisfies associativity and identity laws, making $\mathcal{G}$ a category. Definition (7) satisfies associativity and left-identity laws, making $\mathcal{S}_Z$ a left $\mathcal{G}_Z$-module. □

Prop. 5 is proved by the same "zipping" argument that is used in the deterministic case; see e.g. (McCusker, 1996).

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

Proposition 6 All compositions of the form $R \xrightarrow{\mathcal{F}_Z f} S \xrightarrow{\sigma} T$ or $R \xrightarrow{\mathcal{F}_Z f} S \xrightarrow{\sigma}$ or $R \xrightarrow{\sigma} S \xrightarrow{\mathcal{F}_Z f} T$ are obtained by token-renaming along f. □

It immediately follows that the isomorphisms given by renaming

$$\mathcal{G}_Z(R,S) \times \mathcal{G}_Z(R,T) \cong \mathcal{G}_Z(R, S \uplus T)$$
$$\prod_{i\in I} \mathcal{S}_Z(R \uplus S_i) \cong \mathcal{G}_Z(R, \mathsf{pt}_{i\in I} S_i)$$

are natural in $R \in \mathcal{G}_Z^{\mathsf{op}}$, and so $(\mathcal{G}_Z, \mathcal{S}_Z)$ is a JWA pre-families structure. Moreover, $\mathcal{F}_Z$ preserves finite products on the nose. Finally, we prove Prop. 3 by another zipping argument.

5.5 Enrichment

Let $Z = \{I_o\}_{o\in O}$ be an input signature. We need to make $(\mathcal{G}_Z, \mathcal{S}_Z)$ into a T_Z-enriched JWA judgement model. This resembles the multiplication of T_Z in Sect. 3.1: we define the function $T_Z\mathcal{S}_Z(R) \xrightarrow{\gamma_Z R} \mathcal{S}_Z R$ to map (A, B, C) to (2). It is easy to check that $(\mathcal{S}_Z(R), \gamma_Z R)$ is an T_Z-algebra, and that $\gamma_Z R$ is natural in $R \in \mathcal{G}$. We conclude that $(\mathcal{G}_Z, \mathcal{S}_Z, \gamma_Z)$ is a T_Z-enriched JWA judgement model.

The induced operations $\bigcup$ and input, following the construction in Def. 7, are the same as in Def. 6(2)–(3).

5.6 Type Recursion

Recursive types are modelled following (McCusker, 1996). For arenas R and S, we say that $R \sqsubseteq S$ when for every $r \in R$, both r and all its ancestors are elements of S and $\vdash_R$ is the restriction of $\vdash_S$ to R. We adapt this to arena families: $\{R_i\}_{i\in I} \sqsubseteq \{S_j\}_{j\in J}$ when for each $i \in I$, we have $i \in J$ and $R_i \sqsubseteq S_i$.

We define $\mathcal{E}$ to be the large cpo of countable families of arenas, ordered by $\sqsubseteq$. It is easy to see that the functions

$$\mathcal{E}^I \xrightarrow{\sum_{i\in I}} \mathcal{E} \qquad \mathcal{E}^2 \xrightarrow{\times} \mathcal{E} \qquad \mathcal{E} \xrightarrow{\neg} \mathcal{E}$$

are continuous.

A type context Φ denotes $[\![\Phi]\!] \stackrel{\text{def}}{=} \mathcal{E}^n$, where n is the length of Φ. A type $\Phi \vdash^{\mathsf{type}} A$ denotes a continuous function $[\![\Phi]\!] \xrightarrow{[\![A]\!]} \mathcal{E}$. In particular, if $\Phi, X \vdash^{\mathsf{type}} A$ and $\chi \in \mathcal{E}^{|\Phi|}$ then $[\![\mu \mathtt{X}.A]\!]\chi$ is the least fixpoint of $R \mapsto [\![A]\!](\chi, X \mapsto R)$.

So in the semantics we have an "equirecursive" type:

$$[\![\mu \mathtt{X}.A]\!] = [\![A[\mu \mathtt{X}.A/\mathtt{X}]]\!]$$

We accordingly define $[\![\texttt{fold}\ V]\!]$ to be $[\![V]\!]$, and interpret $\texttt{pm}\ V\ \texttt{as fold x}.\ M$ the same way as $\texttt{let}\ V\ \texttt{be x}.\ M$.

5.7 Statement of Adequacy

Adapting diagram (1) the statement of computational adequacy is as follows. Let Γ be a typing context, denoting the arena family $\{R_i\}_{i\in I}$. The operational semantics gives us a function $\mathsf{JWA}(Z,Y,\Gamma) \xrightarrow{[-]} T_Z\grave{\mathsf{JWA}}(Z,Y,\Gamma)$.

The denotational semantics interprets the judgement $\Gamma \vdash^{\mathsf{n}}$ by the algebra

$$(X,\theta) \stackrel{\text{def}}{=} \prod_{i\in I}(\mathcal{S}_Z R_i, \gamma_Z R_i)$$

Computational adequacy for commands $\Gamma \vdash^{\mathsf{n}} M$ is the commutativity of

$$\begin{array}{ccc} \mathsf{JWA}(Z,Y,\Gamma) & \xrightarrow{[-]} & T_Z\grave{\mathsf{JWA}}(Z,Y,\Gamma) \\ {\scriptstyle [\![-]\!]}\downarrow & & \downarrow{\scriptstyle T_Z[\![-]\!]} \\ X & \xleftarrow[\theta]{} & T_Z X_\Gamma \end{array}$$

This is equivalent to the commutativity of

$$\begin{array}{ccc} \mathsf{JWA}(Z,Y,\Gamma) & \xrightarrow{[-]} & T_Z\grave{\mathsf{JWA}}(Z,Y,\Gamma) \\ {\scriptstyle [\![-]\!]i}\downarrow & & \downarrow{\scriptstyle T_Z([\![-]\!]i)} \\ \mathcal{S}_Z R_i & \xleftarrow[\gamma_Z R_i]{} & T_Z\mathcal{S}_Z R_i \end{array} \tag{8}$$

for every $i \in I$. Explicitly, (8) says that, for any command $\Gamma \vdash^{\mathsf{n}} M$, where $[M] = (A,B,C)$, the strategy

$$\begin{aligned} [\![M]\!]i \stackrel{\text{def}}{=} & (A_{\mathsf{input}} \cup \{ll' \mid lT \in A_{\mathsf{termin}}, [\![T]\!]i = (A',B',C'), l' \in A'\}, \\ & (B \cup \{ll' \mid lT \in A_{\mathsf{termin}}, [\![T]\!]i = (A',B',C'), l' \in B'\}, \\ & (C \cup \{ll' \mid lT \in A_{\mathsf{termin}}, [\![T]\!]i = (A',B',C')l' \in C'\}) \end{aligned}$$

is equal to $[\![M]\!]i$.

6 Determinism and Liveliness

6.1 Determinism

Let $\sigma = (A, B, C)$ be a strategy over Z. (It could be a V-terminable strategy, or a strategy on an arena.) We say that σ is *deterministic* when

- for each Proponent-awaiting play l whose input-awaiting prefixes are all in A, either
  - $l \notin B$ and l has a unique one-place extension in A, or
  - $l \in B$ and has no extension in A
- each infinite trace l whose input-awaiting prefixes are all in A is in C.

Thus a deterministic strategy (A, B, C) is determined by A.

We write

- $T_Z^{\mathsf{det}} V$ for the set of $\sigma \in T_Z V$ that are deterministic
- $\mathcal{S}_Z^{\mathsf{det}} R$ for the set of $\sigma \in \mathcal{S}_Z R$ that are deterministic
- $\mathcal{G}_Z^{\mathsf{det}}(R, S)$ for the set of $f \in \mathcal{G}_Z(R, S)$ that are deterministic at each $b \in \mathsf{rt}\ S$.

Then T_Z^{det} forms a monad on $\mathbf{Set}$, and $(\mathcal{G}_Z^{\mathsf{det}}, \mathcal{S}_Z^{\mathsf{det}}, \gamma_Z^{\mathsf{det}})$ forms a T_Z^{det}-enriched JWA pre-families structure, just like its nondeterministic counterpart. Moreover, the property of determinism is preserved by $\mathsf{input}^o_{i \in I_o}$ for any $o \in O$.

Remark 5 The monad T_Z^{det} can be defined as a "free completely iterative monad" using terminal coaglebras, in the manner of (Aczel et al., 2003; Ghani et al., 2003; Moss, 2001). Explicitly, it maps a set V to $\nu X.(V + R_Z)_\perp$, where R_Z is as defined in Remark 1. □

We obtain

- for every deterministic terminable BLTS $\mathcal{M}$, a function $\mathcal{M} \xrightarrow{[-]^{\mathsf{det}}} T_Z^{\mathsf{det}} \grave{\mathcal{M}}$
- a denotational semantics $[\![-]\!]^{\mathsf{det}}$ of $\mathsf{JWA}(Z, \emptyset)$ in $(\mathcal{G}_Z^{\mathsf{det}}, \mathcal{S}_Z^{\mathsf{det}})$.

6.2 Liveliness

We recall that an erratic signature $Y = \{P_h\}_{h \in H}$ is *lively* when P_h is nonempty for each $h \in H$. For such a signature, the terminable BLTS $\mathcal{L}(Z, Y, \Gamma)$ is *lively*, meaning that each silent state has at least one successor. We define a corresponding notion for strategies, based on (Roscoe, 1998).

Definition 19 Let $\sigma = (A, B, C)$ be a V-terminable strategy, or a strategy on arena R, over input signature Z. We say that σ is *lively* when for every Proponent-awaiting play l whose Opponent-awaiting prefixes are all in A, there is a deterministic strategy starting from l that is contained in σ. □

The property of liveliness is preserved by $\mathsf{input}^o_{i \in I_o}$ for any $o \in O$, and by nonempty union. We write

- $T^+_Z V$ for the set of $\sigma \in T_Z V$ that are lively
- $\mathcal{S}^+_Z R$ for the set of $\mathcal{S} \in \mathcal{S}_Z R$ that are lively
- $\mathcal{G}^+_Z(R, S)$ for the set of $f \in \mathcal{G}_Z(R, S)$ that are lively at each $b \in \mathsf{rt}\ S$.

Then T^+_Z forms a monad on $\mathbf{Set}$, and for any terminable BLTS $\mathcal{M}$ over Z that is lively, we have $\mathcal{M} \xrightarrow{[-]} T^+_Z \grave{\mathcal{M}}$.

As expected, $(\mathcal{G}^+_Z, \mathcal{S}^+_Z, \gamma^+_Z)$ forms a T^+_Z-enriched JWA pre-families structure. So for any input signature Z and lively erratic signature Y, we obtain a model of $\mathsf{JWA}(Z, Y)$ consisting of lively strategies.

The following result shows that liveliness is a sufficiently restrictive constraint.

Proposition 7 Any lively strategy σ on R over Z is a union of a nonempty family of deterministic strategies. □

Proof Suppose l is an infinite trace of σ. For every Opponent-awaiting prefix l' of l, there is a deterministic strategy $\tau(l')$, starting at l, contained in σ. We define $\nu(l)$ to be the deterministic strategy whose finite traces are all the Proponent-awaiting prefixes of l, and, for each l', those finite traces of $\tau(l')$ that disagree with l immediately after l'. Then $\nu(l)$ has l as an infinite trace and is contained in σ. Similarly we can define $\nu(l)$ for each finite trace and divergence of σ. Then σ is the union of $\nu(l)$ as l ranges over finite traces, divergences and infinite traces.

The family is nonempty because σ must have a finite trace or divergence (take a deterministic strategy starting at ϵ contained in σ). □

7 Proving Computational Adequacy

7.1 Weak Adequacy Results

Our desired adequacy theorem can be broken into two parts:

$$\llbracket M \rrbracket i \subseteq [\![M]\!] i \tag{9}$$
$$[\![M]\!] i \subseteq \llbracket M \rrbracket i \tag{10}$$

where $(A, B, C) \subseteq (A', B', C')$ means $A \subseteq A'$ and $B \subseteq B'$ and $C \subseteq C'$.

Before we embark on our proof, we note in this section that there are weak versions of (9)–(10) that are trivial.

We begin with a one-step adequacy result.

Lemma 2 (1) If $\Gamma \vdash^{\mathsf{n}} M$ is silent then $\llbracket M \rrbracket i = \bigcup_{M \rightsquigarrow N} [\![N]\!] i$.
(2) If $\Gamma \vdash^{\mathsf{n}} M$ is o-interactive then $\llbracket M \rrbracket i = \mathsf{input}^{o}_{j \in I_o} [\![M : j]\!] i$.

□

Proof This follows from the categorical structure, which validates all the β-laws. □

Lemma 2(1) tells us that $M \rightsquigarrow M'$ implies $\llbracket M' \rrbracket i \subseteq \llbracket M \rrbracket i$. Hence

$$M \rightsquigarrow^* T \quad \text{implies} \quad [\![T]\!] i \subseteq \llbracket M \rrbracket i \tag{11}$$
$$M \rightsquigarrow^* N \downarrow o \quad \text{implies} \quad \mathsf{input}^{o}_{j \in I_o} [\![N : j]\!] i \subseteq \llbracket M \rrbracket i \tag{12}$$

This immediately gives us a weak version of (9).

Lemma 3 Let $\Gamma \vdash^{\mathsf{n}} M$ be a command. Suppose $[M] = (A, B, C)$.

(1) If $l \in A_{\mathsf{input}}$ then l is a finite trace of $\llbracket M \rrbracket i$.
(2) If $lT \in A_{\mathsf{termin}}$ and $[\![T]\!] i = (A', B', C')$ and $l' \in A'$ (resp. B',C') then ll' is a finite trace (resp. divergence, infinite trace) of $\llbracket M \rrbracket i$.

□

Proof

(1) By induction on l. If l is just o, then this follows (12). If $l = ojl''$, then $M \rightsquigarrow^* N \downarrow o$, where $[N : j] = (A'', B'', C'')$, and $l'' \in A''$. By inductive hypothesis, l'' is a finite trace of $[\![N : j]\!] i$ so by (12) ojl'' is a finite trace of $\llbracket M \rrbracket i$.
(2) Similar induction on l, using (11)–(12).

□

Corollary 8 If $\Gamma \vdash^{\mathsf{n}} M$, then every finite trace of $\llbracket M \rrbracket i$ is a finite trace of $[\![M]\!] i$. □

Likewise, we have a weak version of (10).

Lemma 4 Any finite trace (resp. divergence, infinite trace) l of $\llbracket M \rrbracket i$ is either a

finite trace (resp. divergence, infinite trace) or an extension of a divergence of $[M]$. $\square$

Proof We will construct a sequence $M_0, M_1, \ldots$ of commands in context Γ, and a sequence $l_0 \sqsubseteq l_1 \sqsubseteq \cdots$ of Proponent-awaiting plays [7] that are prefixes of l. For each k, we require $l'_k \stackrel{\text{def}}{=} l \setminus l_k$ (i.e. the unique l' such that $l = l_k l'$) to be a finite trace (resp. divergence, infinite trace) of M_k. For $k = 0$ we set

$$M_0 \stackrel{\text{def}}{=} M \qquad l_0 \stackrel{\text{def}}{=} \epsilon \qquad \text{Hence } l'_0 = l$$

Having constructed M_k and l_k, there are 4 possibilities.

(1) M_k is terminal. Then the sequence ends at k.
(2) M_k is silent. Then by Lemma 2(1), there exists M' such that $M_k \rightsquigarrow M'$ and l'_k is a finite trace (resp. divergence, infinite trace) of M'. Then we define

$$M_{k+1} \stackrel{\text{def}}{=} M' \qquad l_{k+1} \stackrel{\text{def}}{=} l_k \qquad \text{Hence } l'_{k+1} = l'_k$$

(3) M_k is o-interactive and $l'_k = o$. Then the sequence ends at k.
(4) M_k is o-interactive and l'_k is of the form ojl''. Then we define

$$M_{k+1} \stackrel{\text{def}}{=} M_k : j \qquad l_{k+1} \stackrel{\text{def}}{=} l_k oj \qquad \text{Hence } l'_{k+1} = l''$$

If this sequence ends in a terminal command M_K, then $l_K M_K$ is a terminating trace of $[M]$ and we are done.

If it ends in an o-interactive state M_K then $l = l_K o$ is an input-awaiting trace of $[M]$ and we are done.

If it is infinite, define l_{max} to be $\sup_{k\in\mathbb{N}} l_k$, which must be a prefix of l. Since l is finite, l_{max} must be finite, so $l_{\mathsf{max}} = l_K$ for some K and we have

$$M_K \rightsquigarrow M_{K+1} \rightsquigarrow \cdots$$

So l_{max} is a divergence of $[M]$, so l extends a divergence of $[M]$ as required.

In the case that l is an infinite trace of $[\![M]\!]$, it is also possible that l_{max} is infinite, in which case l_{max} is an infinite trace of $[M]$ and we are done. $\square$

For a terminable BLTS $\mathcal{M}$, write $\mathsf{DF}(\mathcal{M})$ for the set of states that are *divergence-free*, i.e. have no divergences. We have adequacy for deterministic, divergence-free commands. For Γ denoting $\{R_i\}_{i\in I}$, this amounts to the commutativity of the

[7] Just output and input moves, no arena moves

following variant of (8), for each $i \in I$.

$$\begin{array}{ccc} \mathsf{DF}(\mathsf{JWA}(Z,\emptyset,\Gamma)) & \xrightarrow{[-]^{\det}} & T_Z^{\det}\grave{\mathsf{JWA}}(Z,\emptyset,\Gamma) \\ {\scriptstyle [\![-]\!]^{\det} i}\downarrow & & \downarrow{\scriptstyle T_Z^{\det}[\![-]\!]^{\det} i} \\ \mathcal{S}_Z^{\det} R_i & \xleftarrow[\gamma_Z^{\det} R_i]{} & T_Z^{\det}\mathcal{S}_Z^{\det} R_i \end{array} \tag{13}$$

To see this, suppose $M \in \mathsf{DF}(\mathsf{JWA}(Z,\emptyset,\Gamma))$. Write $[\![M]\!]^{\det} i = (A,B,C)$ and $[\![M]\!]^{\det} i = (A',B',C')$. These are deterministic, so to prove them equal, it suffices to prove $A = A'$. Lemma 8 tells us $A' \subseteq A$. Since M is divergence-free, Lemma 4 implies $A \subseteq A'$.

7.2 Relating Enriched Models

Our proof is going to be based on relating two JWA pre-families structures enriched in different monads. We set up the abstract structure first.

Let T and T' be monads on $\mathbf{Set}$, and let $T \xrightarrow{\delta} T'$ be a monad morphism.

A *mapping across* δ from a T-enriched JWA pre-families structure $(\mathcal{G},\mathcal{S},\gamma)$ to a T'-enriched JWA pre-families structure $(\mathcal{G}',\mathcal{S}',\gamma')$ with the same objects and object structure is a collection of functions

$$\begin{array}{rl} \mathcal{G}(A,B) \xrightarrow{\epsilon(A,B)} \mathcal{G}'(A,B) & \text{for all } A,B \in \mathsf{ob}\,\mathcal{G} \\ \mathcal{S}A \xrightarrow{\epsilon A} \mathcal{S}'A & \text{for all } A \in \mathsf{ob}\,\mathcal{G} \end{array}$$

preserving identity, both kinds of composition, product structure and $\neg$ structure, such that, for every $A \in \mathsf{ob}\,\mathcal{G}$, the following commutes.

$$\begin{array}{ccc} T\mathcal{S}A & \xrightarrow{\delta\epsilon A} & T'\mathcal{S}'A \\ {\scriptstyle \gamma A}\downarrow & & \downarrow{\scriptstyle \gamma' A} \\ \mathcal{S}A & \xrightarrow{\epsilon A} & \mathcal{S}'A \end{array} \tag{14}$$

Remark 6 More abstractly, writing $\underline{\mathcal{S}} = (\mathcal{S},\gamma)$ and $\underline{\mathcal{S}'} = (\mathcal{S}',\gamma')$, a mapping across δ can be defined to be

- an identity-on-objects finite-product-preserving functor $\mathcal{G} \xrightarrow{\epsilon_0} \mathcal{G}'$
- a natural transformation $\underline{\mathcal{S}} \xrightarrow{\epsilon_1} \mathbf{Set}^{\delta}\underline{\mathcal{S}'}\epsilon_0$ in $[\mathcal{G}^{\mathsf{op}}, \mathbf{Set}^T]$

preserving $\neg$ structure. Here $\mathbf{Set}^{T'} \xrightarrow{\mathbf{Set}^{\delta}} \mathbf{Set}^T$ is the functor mapping a T'-algebra (X,θ) to $(X,(\delta_X;\theta))$. □

Suppose that we have an input signature $Z = \{I_o\}_{o \in O}$. An *input signature embedding* into Z consists of a set N and and an injection $N \xrightarrow{\iota} O$. Given such an embedding, we define the input signature $\iota^{-1}Z$ to be $\{I_{\iota(n)}\}_{n \in N}$.

Our aim is to define

- a monad morphism $T_Z^{\mathsf{det}} \xrightarrow{\delta_\iota} T_{\iota^{-1}Z}$
- a mapping $(\mathcal{G}_Z^{\mathsf{det}}, \mathcal{S}_Z^{\mathsf{det}}, \gamma_Z^{\mathsf{det}}) \xrightarrow{\epsilon_\iota} (\mathcal{G}_{\iota^{-1}Z}, \mathcal{S}_{\iota^{-1}Z}, \gamma_{\iota^{-1}Z})$ across δ_ι.

Thus we have to convert deterministic strategies over Z into nondeterministic strategies over $\iota^{-1}Z$. We do this by converting the input^o operators, where $o \in O \setminus \iota(N)$, into erratic operators. This is called *ι-hiding*.

Remark 7 In fact, ι-hiding could be defined on *all* strategies over Z, not just deterministic ones. But that is not needed for our adequacy proof. □

Let l be a V-terminable play over Z, or a play on arena R over Z. We define a play $\mathsf{Hide}_\iota l$ over $\iota^{-1}Z$, the *ι-hiding* of l, by removing from l every o-output move, where $o \in O \setminus \iota(N)$, and every input move that follows such an output move. Also, we replace every output move $\iota(n)$, for $n \in N$, by n.

There are several possibilities for l, listed as follows.

- l awaits $\iota(n)$-input, for $n \in N$, and $\mathsf{Hide}_\iota l$ awaits n-input. (We say that l *awaits ι-visible input*.)
- (For a V-terminable play) l is terminating, and so is $\mathsf{Hide}_\iota l$.
- (For a play on an arena) l awaits arena Opponent, and so is $\mathsf{Hide}_\iota l$.
- l either awaits o-input, for $o \in O \setminus \iota(N)$, or awaits Proponent, and $\mathsf{Hide}_\iota l$ awaits Proponent.
- l is infinite, and $\mathsf{Hide}_\iota l$ awaits Proponent. (We say that l is *ι-starved*.)
- l is infinite, and so is $\mathsf{Hide}_\iota l$. (We say that l is *ι-infinite*.)

Let $\sigma = (A, B, C)$ be a deterministic strategy, either V-terminable or on an arena, over Z. The *ι-hiding* of σ, written $\mathsf{Hide}_\iota \sigma$, is the strategy over $\iota^{-1}Z$ defined as follows.

finite traces(1) the ι-hiding of every $l \in A$ that awaits ι-visible input
finite traces (2) the ι-hiding of every $l \in A$ that is terminating / awaiting arena Opponent
divergences (1) the ι-hiding of every $l \in B$
divergences (2) the ι-hiding of every $l \in C$ that is ι-starved
infinite traces the ι-hiding of every $l \in C$ that is ι-infinite.

Remark 8 An input signature embedding $N \xrightarrow{\iota} O$ is *lively* when for each $o \in O \setminus \iota(N)$, the set I_o is nonempty. If ι is lively, then ι-hiding preserves liveliness of strategies. □

We thus have functions

$$T_Z^{\mathsf{det}} V \xrightarrow{\delta_\iota V} T_{\iota^{-1}Z} V \qquad \text{for every set } V$$

$$\mathcal{S}_Z^{\mathsf{det}} R \xrightarrow{\epsilon_\iota R} \mathcal{S}_{\iota^{-1}Z} R \qquad \text{for every arena } R$$

$$\mathcal{G}_Z^{\mathsf{det}}(R,S) \xrightarrow{\epsilon_\iota(R,S)} \mathcal{G}_{\iota^{-1}Z}(R,S) \qquad \text{for arenas } R, S$$

where the first two are just Hide_ι and the third maps $R \xrightarrow{f} S$ to $\lambda b \in \mathsf{rt}\, S.\, (\mathsf{Hide}_\iota(f_b))$. It can be verified that δ_ι is a monad morphism and ϵ_ι is a mapping across δ_ι, as required. The only non-trivial part is proving that ϵ_ι preserves composition; this is proved by a zipping argument in Sect. 7.4. We also have

$$\mathsf{Hide}_\iota(\mathsf{input}^{\iota(n)}_{i\in I_o}\sigma_i) = \mathsf{input}^{n}_{i\in I_o}\mathsf{Hide}_\iota\sigma_i \quad \text{for } n \in N \tag{15}$$

$$\mathsf{Hide}_\iota(\mathsf{input}^{o}_{i\in I_o}\sigma_i) = \bigcup_{i\in I_o} \mathsf{Hide}_\iota\sigma_i \quad \text{for } o \in O \setminus \iota(N) \tag{16}$$

Since Hide_ι commutes with renaming along a $\mathbf{TokRen}$-morphism $R \xrightarrow{f} S$, the following commutes:

$$\begin{array}{ccc} \mathbf{TokRen}^{\mathsf{op}} & \xrightarrow{\mathcal{F}_Z^{\mathrm{det}}} & \mathcal{G}_Z^{\mathrm{det}} \\ & \searrow^{\mathcal{F}_{\iota^{-1}(Z)}} & \downarrow{\scriptstyle \epsilon_\iota} \\ & & \mathcal{G}_{\iota^{-1}Z} \end{array}$$

7.4 Hiding Preserves Composition

Let $N \xrightarrow{\iota} O$ be an input signature embedding into $Z = \{I_o\}_{o\in O}$. We wish to show that ι-hiding preserves composition; specifically, that for arenas R, S, T we have

$$\mathsf{Hide}_\iota(\sigma \circledast_{R,S,T} \tau) = (\mathsf{Hide}_\iota\sigma) \circledast_{R,S,T} (\mathsf{Hide}_\iota\tau) \tag{17}$$

for any $\sigma \in \mathcal{G}_Z(R,S)$ and $\tau \in \mathcal{S}_Z(S \uplus T)$. This is proved using the same kind of "zipping" argument that is used to prove associativity. Though we have omitted the other zipping proofs, we give this one in detail.

Define A to be the pointed cpo of plays over $\iota^{-1}Z$ on $R \uplus T$, ordered by extension. Define B to be the pointed cpo of interaction sequences over 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 $\iota^{-1}Z$ on R, S, T
- v associates, to $q \in \mathsf{inners(u)}$ in u, a play $v(q)$ over Z on the arena of q whose ι-hiding is the q-thread of u.

The ordering makes $(u, v) \leqslant (u', v')$ when u is a prefix of u' and, for every $q \in \mathsf{inners(u)}$, 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 $q \in \mathsf{inners(u)}$, the ι-hiding of $v(q)$ awaits arena-Opponent and so $v(q)$ awaits arena-Opponent. Hence $u < u'$ (because $v(q) < v'(q)$ implies $u < u'$).

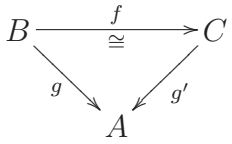
(2) u awaits n-input (where $n \in N$) in thread l; then the ι-hiding of $v(l)$ awaits n-input, so $v(l)$ awaits $\iota(n)$-input; and for each inner thread-name $q \in \mathsf{inners(u)} \setminus \{l\}$, the ι-hiding of $v(q)$ awaits arena-Opponent so $v(q)$ awaits arena-Opponent. Hence $u < u'$ (because $v(q) < v'(q)$ implies $u < u'$).

(3) u is infinite; then for each $q \in \mathsf{inners(u)}$, the ι-hiding of $v(q)$ 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-Proponent; then for each $q \in \mathsf{inners(u)} \setminus \{l\}$, the ι-hiding of $v(q)$ awaits arena-Opponent and so $v(q)$ awaits arena-Opponent. Hence $v(l) < v'(l)$ (because $u < u'$ implies $v(l) < v'(l)$, and $v(q) < v'(q)$ implies $u < u'$ if $q \in \mathsf{inners(u)} \setminus \{l\}$). So $v(l)$ is finite, and since the ι-hiding of $v(l)$ awaits Proponent, $v(l)$ either awaits Proponent or awaits o-input for some $o \in O \setminus \iota(N)$.

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

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 Proponent-move in some thread $l \in \mathsf{inners(u)}$. Hence $(\mathsf{Hide}_\iota v(l))(n \curvearrowleft r) \sqsubseteq \mathsf{Hide}_\iota v'(l)$, 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 $q \in \mathsf{inners(u)} \setminus \{l\}$. 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

$$s_0 < s_1 < s_2 < \cdots \in B \tag{18}$$

such that s_i is the unique B-element of length i whose f-image is $\leqslant (u, v)$. This is defined by induction: $s_0 = \epsilon$, 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 (18) ends in s_n, then $f(s_n) = (u, v)$. If (18) 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 (18), so $s' \leqslant s$, and, since f is strictly monotonic, $s' = s$. Thus f is a poset isomorphism.

Now suppose we are given $\sigma \in \mathcal{G}_Z(R, S)$ and $\tau \in \mathcal{S}_Z(S \uplus T)$. Let t be a Proponent-awaiting play over $\iota^{-1}Z$ on $R \uplus T$. Then t is a divergence of $\epsilon_\iota(\sigma \circledast \tau)$ iff $t = g(s)$, for some $s \in B$ such that (condition 1)

- s awaits l-Proponent, its l-thread is a divergence of $l(\sigma, \tau)$, and the q-thread of s is a finite trace of $q(\sigma, \tau)$ for each $q \in \mathsf{inners(s)} \setminus \{l\}$, 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-Proponent, $v(l)$ is a divergence of $l(\sigma, \tau)$, and $v(q)$ is a finite trace of $q(\sigma, \tau)$ for each $q \in \mathsf{inners(u)} \setminus \{l\}$
- u awaits l-Player, $v(l)$ is an infinite trace of $l(\sigma, \tau)$, and $q(l)$ is a finite trace of $q(\sigma, \tau)$ for each $q \in \mathsf{inners(u)} \setminus \{l\}$
- u is infinite, and $v(q)$ is a finite trace or infinite trace of $q(\sigma, \tau)$ for each $q \in \mathsf{inners(u)}$.

Any $s \in B$ satisfies condition 1 iff $f(s)$ satisfies condition 2, so the two sides of (17) have the same divergences. By a similar but easier argument, they have the same finite traces and infinite traces.

7.5 Unhidings

In the next section, we shall look at an *unhiding transform* from a nondeterministic calculus to a deterministic one. In this section, we look at the essential features such a transform ought to have.

Definition 20 Let $\mathcal{M}$ be a BLTS over $\iota^{-1}Z$ and let $\mathcal{M}'$ be a deterministic BLTS over Z. A function $\mathcal{M} \xrightarrow{f} \mathcal{M}'$ is an *unhiding* when, for every state $d \in \mathcal{M}$, we have the following.

- If d is terminal then $f(d)$ is terminal.
- If d is n-interactive, for $n \in N$, then $f(d)$ is $\iota(n)$-interactive, and $f(d) : i = f(d : i)$ for all $i \in I_{\iota(n)}$.
- If d is silent then there exists (necessarily unique) $e \in \mathcal{M}'$ and $o \in O \setminus \iota(L)$ such that $f(d) \rightsquigarrow^* e \downarrow o$, and furthermore

$$\{e : i \mid i \in I_{o'}\} = \{f(d') \mid d \to d'\}$$

□

Lemma 5 Let $\mathcal{M}$ be a BLTS over $\iota^{-1}Z$ and let $\mathcal{M}'$ be a deterministic BLTS over Z. Let $\mathcal{M} \xrightarrow{f} \mathcal{M}'$ be an unhiding, so it restricts to a function $\grave{\mathcal{M}} \xrightarrow{f} \grave{\mathcal{M}}'$. Then the range of f is contained in $\mathsf{DF}(\mathcal{M}')$, and the following diagram commutes.

$$\begin{array}{ccc} \mathcal{M} & \xrightarrow{[-]} & T_{\iota^{-1}Z}\grave{\mathcal{M}} \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \delta_\iota f} \\ \mathsf{DF}(\mathcal{M}') & \xrightarrow[[-]]{} & T_Z^{\mathsf{det}}\grave{\mathcal{M}}' \end{array}$$

□

7.6 Adequacy Via Unhiding

Given an input signature $Z = \{I_o\}_{o \in O}$ and an erratic signature $Y = \{P_h\}_{h \in H}$, we want to prove the adequacy of $\mathsf{JWA}(Z, Y)$, using the tools we have developed.

We define the input signature Z' to be Z extended with Y and a unary ✓ operator. Formally it is $\{I'_o\}_{o \in O'}$ defined as follows. The indexing set is $O' \stackrel{\text{def}}{=} O + H + \{\checkmark\}$. We write $O \xrightarrow{\iota} O'$ and $H \xrightarrow{\iota'} O'$ for the embeddings. We define $I'_{\iota(o)}$ to be I_o (for $o \in O$), we define $I'_{\iota'(h)}$ to be P_h (for $h \in H$), and we define $I'_\checkmark$ to be singleton.

We note that ι is an input signature embedding into Z', giving $\iota^{-1}Z' = Z$.

$\Gamma \vdash^{\mathsf{n}} M$	$\Gamma \vdash^{\mathsf{n}} u(M)$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ N$	$\mathtt{let}\ u(V)\ \mathtt{be}\ \mathtt{x}.\ \checkmark.u(N)$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i\in I}$	$\mathtt{pm}\ u(V)\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\checkmark.u(M_i)\}_{i\in I}$
$\mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M$	$\mathtt{pm}\ u(V)\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\checkmark.u(M)$
VW	$u(V)u(W)$
$\mathtt{pm}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.M$	$\mathtt{pm}\ u(V)\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\checkmark.u(M)$
$\mathtt{choose}^h\{M_p\}_{p\in P_h}$	$\mathtt{input}^{\iota'(h)}\{u(M_p)\}_{p\in P_h}$
$\mathtt{input}^o\{M_i\}_{i\in I_o}$	$\mathtt{input}^{\iota(o)}\{u(M_i)\}_{i\in I_o}$

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

Fig. 6. The unhiding transform

Remark 9 If the erratic signature Y is lively, then the input signature embedding ι is lively in the sense of Remark 8. □

We will define two transforms. The *hiding transform* h is from $\mathsf{JWA}(Z', \emptyset)$ to $\mathsf{JWA}(Z, Y)$. This consists of

- removing every occurrence of $\checkmark$
- replacing every occurrence of $\mathsf{input}^{\iota'(h)}$, where $h \in H$, by choose^h
- replacing every occurrence of $\mathsf{input}^{\iota(o)}$, where $o \in O$, by input^o.

This transform exactly corresponds to ι-hiding on the semantics.

Lemma 6 Let M be a JWA term (command or value). Then $[\![h(M)]\!] = \mathsf{Hide}_\iota[\![M]\!]$. □

Proof Straightforward induction, using the fact that Hide_ι preserves all categorical structure, and (15)–(16). □

We next define an *unhiding transform* u from $\mathsf{JWA}(Z, Y)$ to $\mathsf{JWA}(Z', \emptyset)$. This is shown in Fig. 6.

Lemma 7 For any term M (command or value) of $\mathsf{JWA}(Z, Y)$, we have $h(u(M)) = M$ (syntactic identity). □

The syntactic properties of unhiding are as follows. The following lemma gives the operational properties of the unhiding transform.

Lemma 8 (1) The unhiding transformation preserves renaming and substitution. In particular, $u(M[V/\mathtt{x}]) = u(M)[u(V)/\mathtt{x}]$

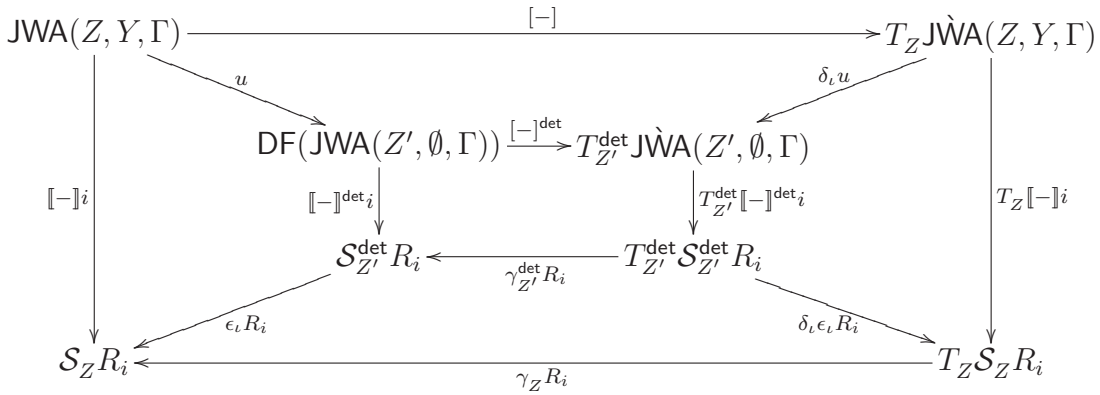
(2) For any command $\Gamma \vdash^{\mathsf{n}} M$ in $\mathsf{JWA}(Z, Y)$,

- if M is terminal then $u(M)$ is terminal
- if M is silent and not of the form $\mathtt{choose}^{h}_{p \in P_h} M_p$, then M has a unique successor M' and $u(M)$ is silent with unique successor $\checkmark.u(M')$.

□

Corollary 9 u is an unhiding from $\mathsf{JWA}(Z, Y)$ to $\mathsf{JWA}(Z', \emptyset)$. □

We now have everything in place to prove (8), by means of the diagram



(19)

The top part of (19) is an instance of Lemma 5. The central part is an instance of diagram (13). As stated in Sect. 7.3, ϵ_ι is a mapping across δ_ι, in particular satisfying (14), which gives us the lower part.

The right part of (19) is obtained by applying δ_ι horizontally to the left part, and restricting to terminal commands. So only the left part remains to be proved. It is

given by

$$\begin{array}{ccccc}
\mathsf{JWA}(Z,Y,\Gamma) & & \xrightarrow{\quad u \quad} & & \\
\Big\downarrow{\scriptstyle \mathsf{id}} & & & & \\
\mathsf{JWA}(Z,Y,\Gamma) & \xleftarrow{\quad h \quad} & \mathsf{DF}(\mathsf{JWA}(Z',\emptyset,\Gamma)) & & \\
\Big\downarrow{\scriptstyle [\![-]\!]i} & & \Big\downarrow{\scriptstyle [\![-]\!]^{\mathsf{det}} i} & & \\
\mathcal{S}_Z R_i & \xleftarrow{\quad \epsilon_\iota R_i \quad} & \mathcal{S}^{\mathsf{det}}_{Z'} R_i & &
\end{array} \tag{20}$$

where the top part of (20) is Lemma 7 and the bottom part is Lemma 6. Alternatively the left part of (19) can be proved directly by induction, avoiding the need to define h.

7.7 Empty Signatures

We briefly discuss what this adequacy argument reduces to in the case of a language that has no I/O, i.e. where the input signature Z is empty. In particular

- the monad $T_\emptyset$ is $V \mapsto \mathcal{P}(V_\perp)$
- the monad $T^+_\emptyset$ is $V \mapsto \mathcal{P}^+(V_\perp)$
- the moand $T^{\mathsf{det}}_\emptyset$ is $V \mapsto V_\perp$

If, moreover, the erratic signature Y is empty, so that the language is deterministic as in McCusker (1996), then Z' consists of a single unary operator $\checkmark$. So $T^{\mathsf{det}}_{Z'}$ is $V \mapsto \mathbb{N} \times V + \{\infty\}$. In this situation, the unhiding transform merely adds a $\checkmark$ for each transition, ensuring that the translation of every term is non-divergent.

This gives a considerably simpler adequacy proof than that provided in (McCusker, 1996), which uses the relational technique of (Pitts, 1996). But each method has its advantages: only the unhiding proof works for models of infinite trace equivalence, and only the relational proof works for domain models.

8 The meaning of a non-lively language

8.1 Omni-errors

A valid implementation of an imperative language must execute each primitive command, such as `print` or `choose`, within a finite time. An implementation that tarries forever while executing a command is incorrect. Therefore, a language built

from a non-lively erratic signature cannot be implemented, as it contains a command "erratically choose an element of the empty set", which cannot be executed.

Nevertheless, such a language *can* be given operational meaning, using the concept of *omni-errors*, as we now explain.

Take any programming language, e.g. Java. Let U be a set, whose elements we call "omni-errors". Define Java_U to be the following nondeterministic language:

- the syntax is that of Java
- the operational semantics is that of Java, except that any program, at any time, is allowed to throw any $u \in U$, i.e. to output u and terminate.

Note that $\mathsf{Java}_\emptyset$ is Java.

Since omni-errors can be thrown by *any* term, they do not affect (any notion of) observational equivalence. For this reason, the denotational theory of Java_U is exactly the same as that of Java. So the set U is denotationally immaterial.

If U is nonempty, then the extension of Java_U with an empty choice command can be given operational meaning: to execute empty choice, simply choose some omni-error $u \in U$ and throw it. We accordingly say that an erratic signature Y (or a terminable BLTS, or a strategy, or an input signature embedding) is *lively with respect to* a set U of omni-errors when either Y is lively or U is nonempty. This means that a language that has *both* omni-errors provided by U and erratic choice provided by Y is operationally meaningful.

8.2 Finite traces and infinite trace equivalence

Recall the calculus $\mathcal{L}(\mathcal{A}, Y)$ from Sect. 2.1. We write $\mathcal{L}_U(\mathcal{A}, Y)$ for the extension with a set U of omni-errors.

For a closed term M in $\mathcal{L}(\mathcal{A}, Y)$, let A, B and C be the sets of finite traces, divergences, and infinite traces of M, respectively. We define

$$[M] \stackrel{\text{def}}{=} (A, B, C)$$
$$[M]_U \stackrel{\text{def}}{=} (A \times U, B, C) \text{ for any set } U \text{ wrt which } Y \text{ is lively}$$

If Y is lively wrt U, then $[M]_U$ is the set of possible behaviours of M in the (operationally meaningful) calculus $\mathcal{L}_U(\mathcal{A}, Y)$, because M can

- print some $l \in A$, then throw some $u \in U$
- print some $l \in B$, then diverge
- print some $l \in C$.

Proposition 10 Let Y be an erratic signature lively wrt a set U of omni-errors. Then the kernel of $[-]_U$ and the kernel of $[-]$, as equivalence relations on closed terms in $\mathcal{L}(\mathcal{A}, Y)$, are the same. □

This justifies defining "infinite trace equivalence" on $\mathcal{L}(\mathcal{A}, Y)$ to be the kernel of $[-]$, as we do in Sect. 2.1.

9 Further Work: General References

The adequacy proof above should be adapted to general references (Abramsky et al., 1998), but this seems likely to go through smoothly. Furthermore, when general references are added to JWA, the results of (Abramsky et al., 1998) ought to give definability and full abstraction results.

Adapting Prop. 7, it appears that any lively strategy is definable in the presence of continuum choice. (This assumes that the input signature Z is countable.) A variant for general (non-lively) strategies should be straightforward.

For full abstraction, we conjecture that distinct strategies over Z can be distinguished by a strategy over $Z + \{\checkmark\}$, where $\checkmark$ is a unary operator. However, the semantics for a fixed input signature Z might not be fully abstract.

References

Abramsky, S., 1983. On semantic foundations for applicative multiprogramming. In: Díaz, J. (Ed.), Automata, Languages and Programming, 10th Colloquium. Vol. 154 of LNCS.

Abramsky, S., Honda, K., McCusker, G., 1998. A fully abstract game semantics for general references. In: Proceedings, 13th Annual IEEE Symposium on Logic in Computer Science. pp. 334–344.

Abramsky, S., McCusker, G., 1998. Call-by-value games. In: Nielsen, M., Thomas, W. (Eds.), Computer Science Logic: 11th International Workshop Proceedings. LNCS. Springer, pp. 1–17.

Aczel, P., Adámek, J., Milius, S., Velebil, J., 2003. Infinite trees and completely iterative theories: a coalgebraic view. Theoretical Computer Science 300 (1-3), 1–45.

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

Broy, M., 1986. A theory for nondeterminism, parallelism, communication, and concurrency. Theoretical Computer Science 45, 1–61.

Cattani, G. L., Winskel, G., 2003. Presheaf models for CCS-like languages. Theoretical Computer Science 300 (1-3), 47–89.

Escardó, M., 1998. A metric model of PCF, unpublished research note.
Ghani, N., Lüth, C., Marchi, F. D., Power, J., 2003. Dualising initial algebras. Mathematical Structures in Computer Science 13 (2), 349–370.
Harmer, R., McCusker, G., 1999. A fully abstract game semantics for finite nondeterminism. In: 14th Symposium on Logic in Comp. Sci. IEEE.
Hasegawa, M., 1997. Models of sharing graphs :–a categorical semantics of let and letrec. Ph.D. thesis, University of Edinburgh.
Hyland, J. M. E., Ong, C.-H. L., 2000. On full abstraction for PCF: I, II, and III. Information and Computation 163 (2).
Jonsson, B., 1994. A fully abstract trace model for dataflow and asynchronous networks. Distributed Computing 7 (4).
Lassen, S. B., Levy, P. B., 2007. Typed normal form bisimulation. In: Duparc, J., Henzinger, T. (Eds.), Proc., 23rd Conf. on Comp. Sci. and Logic. Vol. 4646 of LNCS.
Levy, P. B., 2004a. Call-By-Push-Value. A Functional/Imperative Synthesis. Semantic Struct. in Computation. Springer.
Levy, P. B., April 2004b. Infinite trace semantics, Proc., 2nd APPSEM II Workshop, Tallinn, Estonia.
Levy, P. B., 2005. Adjunction models for call-by-push-value with stacks. Theory and Applications of Categories 14, 75–110.
Levy, P. B., 2006a. Call-by-push-value: Decomposing call-by-value and call-by-name. Higher-Order and Symbolic Computation 19 (4), 377–414.
Levy, P. B., 2006b. Infinite trace equivalence. In: Proc., 21st Ann. Conf. in Mathematical Foundations of Comp. Sci., Birmingham, UK, 2005. No. 155 in ENTCS.
McCusker, G., 1996. Games and full abstraction for a functional metalanguage with recursive types. Ph.D. thesis, University of London.
Møgelberg, R., Simpson, A., 2007. Relational parametricity for computational effects. In: Proceedings, 22nd IEEE Symposium in Logic in Computer Science. IEEE Computer Society, pp. 346–355.
Moggi, E., 1991. Notions of computation and monads. Information and Computation 93.
Moss, L. S., 2001. Parametric corecursion. Theoretical Computer Science 260 (1-2), 139–163.
Nickau, H., 1996. Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality. Shaker-Verlag, diss., Universität Gesamthochschule Siegen.
Panangaden, P., Russell, J. R., 1989. A category-theoretic semantics for unbounded indeterminacy. In: Proceedings, 5th Conference on Mathematical Foundations of Programming Semantics, New Orleans. Vol. 442 of LNCS. pp. 319–332.
Pitts, A. M., 1996. Relational properties of domains. Information and Computation 127.
Plotkin, G., 1983. Domains, prepared by Y. Kashiwagi, H. Kondoh and T. Hagino.
Plotkin, G., Power, J., 2002. Notions of computation determine monads. In: Proceedings, Foundations of Software Science and Computation Structures, 2002. Vol. 2303 of LNCS. Springer, pp. 342–356.

Plotkin, G. D., Power, A. J., 2001. Adequacy for algebraic effects. LNCS 2030.
Plotkin, G. D., Power, A. J., 2003. Algebraic operations and generic effects. Applied Categorical Structures 11 (1), 69–94.
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”.
Streicher, T., Reus, B., 1998. Classical logic, continuation semantics and abstract machines. Journal of Functional Programming 8 (6), 543–572.