Typed Normal Form Bisimulation for Parametric Polymorphism

Soren B. Lassen
Google, Inc.
soren@google.com

Paul Blain Levy
University of Birmingham, U.K.
pbl@cs.bham.ac.uk

Abstract

This paper presents a new bisimulation theory for parametric polymorphism which enables straightforward coinductive proofs of program equivalences involving existential types. The theory is an instance of typed normal form bisimulation and demonstrates the power of this recent framework for modeling typed lambda calculi as labelled transition systems.

We develop our theory for a continuation-passing style calculus, Jump-With-Argument, where normal form bisimulation takes a simple form. We equip the calculus with both existential and recursive types. An "ultimate pattern matching theorem" enables us to define bisimilarity and we show it to be a congruence. We apply our theory to proving program equivalences, type isomorphisms and genericity.

1 Introduction

Parametric polymorphism is a captivating programming language concept, both because of its important role in modern programming language theory and practice, and because of its fascinating relational parametricity properties. The need for good semantic theories for reasoning about parametric polymorphism is evident and this has been the subject of many research efforts, since Reynolds relational parametricity result for the pure polymorphic λ-calculus [22]. Theories for polymorphic calculi with fixed point recursion or recursive types have encountered a number of difficulties, either complex meta-theories, problems with type recursion, or weak reasoning principles.

The theory we develop in this paper is an instance of typed normal form bisimulation, a framework which was recently introduced for a monomorphic, recursively typed calculus [14]. Given a typed λ-calculus with an abstract machine semantics, the development of its normal form bisimulation theory proceeds in four steps.

1. An ultimate pattern matching theorem uniquely decomposes a value into an ultimate value pattern and a "filling" of function value sub-terms.
2. Abstract machine states are interpreted as states in a labelled transition system (LTS) with ultimate patterns as labels.
3. Bisimilarity between LTS states defines a normal form bisimulation congruence relation on terms.
4. The congruence property follows from a substitution lemma about bisimilarity between LTS states which, in turn, is proved by bisimulation.

The LTS is bi-partite: a state is either active or passive, and a labelled transition is either an output transition from an active to a passive state or an input transition from a passive to an active state. We think of labelled transitions as alternating moves in a game between a program and its environment. Each input (output) move corresponds to the invocation of a function exported by the program (environment). The label identifies the exported function and also describes the function argument, as an ultimate value pattern.

Following [14], we develop our theory for the continuation-passing style calculus Jump-With-Argument (JWA). Because functions never return, the LTS takes a simple form, with only one input and one output transition. This simplifies the exposition and clarifies the underlying concepts.

Although omitted in this presentation, we note that our work generalizes to richer λ-calculi. Direct style typed λ-calculi translate into JWA and the translations together with the JWA normal form bisimulation theory induce normal form bisimilation theories for the direct style source calculi, analogously to the way CPS transforms preserve and reflect normal form bisimilarity in the untyped λ-calculus [12, 13]. For typed calculi, these translations and induced theories factor through a typed normal form bisimulation theory for the Call-By-Push-Value calculus and the stack-passing transform into JWA [16].

Our theory is not fully abstract for JWA, for the same reason (same counter examples) as in the monomorphic case [14]. We expect that our treatment of polymorphism can be combined with state in the style of [15, 10] and then, if we

extend the JWA calculus with state, we conjecture that our theory becomes fully abstract.

1.1 Contributions

Parametric polymorphism is a very useful and non-trivial addition to the growing set of normal form bisimulation theories, thus both attesting to the power and scope of this semantic framework and contributing towards the goal of scaling normal form bisimulation to feature rich higher-order typed programming languages. Compared to the monomorphic typed normal form bisimulation theory in [14], our key discovery is the definition of ultimate patterns for existential types.

In its own right, the bisimulation theory in this paper is a novel semantic theory for second-order typed λ-calculi with an elementary meta-theory and a new, powerful and straightforward bisimulation proof rule for reasoning about program equivalences involving existential types, which we illustrate with several example proofs. The theory is robust in that it supports recursive types and we expect that, like other normal form bisimulation theories, it can be combined with computational effects in the style of [15, 10].

Our LTS brings out connections with game semantics, by design [15], and it seems likely that our theory could lead to a Hyland-Ong style model of polymorphism, fully abstract in the presence of state—by analogy with [1, 10].

1.2 Related Work

Our work builds on the LTS and typed normal form bisimulation theory for a monomorphic recursively typed CPS calculus in [14]. (See *op.cit.* for a survey of earlier work on normal form bisimulation for untyped λ-calculi.) Laird's LTS for a direct style typed calculus [10] is very similar and it also influenced the presentation of our LTS in the present paper. Laird derives a trace semantics from his LTS, whereas we derive a bisimulation semantics. Because our calculus is deterministic, the two coincide (in the absence of Laird's completeness constraint on traces, which does not apply to a CPS calculus). We prefer bisimilarity to trace equivalence because of the convenience of the associated co-induction proof principle for articulating proofs of bisimilarity, namely by exhibiting a bisimulation.

Pitts' operationally based logical relations [20] is a different syntactic theory for parametric polymorphism with powerful relational proof principles comparable to normal form bisimulation. The theory is fully abstract for a state-less calculus, contrary to our theory, but recursive types present a difficulty and are not covered. This limits the scope of the theory and has motivated work on alternative operationally based approaches, including our work.

Ahmed's step-indexed syntactic logical relations [3] solve the difficulty of recursive types by stratifying the relations by the number of steps available for future evaluation. They form a fully abstract syntactic model for a state-less polymorphic calculus with recursive types.

Gordon's applicative bisimulation theory [7] and Sumii and Pierce's relation-sets bisimulation theory [24] for parametric polymorphism are two other operationally based theories with recursive types, relational congruence proofs, and straightforward bisimulation proof rules. Both theories are fully abstract. Their bisimulation proof rules are weaker than our normal form bisimulation, because their proof obligations involve quantification over all closed arguments to functions and all closed types to type abstractions, whereas normal form bisimulation evaluates related functions and type abstractions by effectively applying them to fresh identifiers or type identifiers. Because of this difference, normal form bisimulation proofs are generally more direct without any need for auxiliary "bisimulation up-to" techniques.

Many of the ideas we use to model second order types in our LTS appear in the LTSs for polymorphically typed π-calculi in [19, 4] but the combination with recursive, sum, product, and function types via ultimate patterns is new.

Whereas typed normal form bisimulation for monomorphic types [14] appears to be closely related to the game semantics of Hyland and Ong [9], our polymorphic extension seems not to correspond to existing game models of polymorphism. They are either highly intensional [8, 5] or in the Abramsky-Jagadeesan-Malacaria (AJM) style of game semantics [2, 18].

Outline Section 2 introduces JWA and its operational semantics with some examples to illustrate its continuation-passing style. Section 3 introduces ultimate patterns and states the ultimate pattern matching theorem. Section 4 defines a LTS. We develop the resulting bisimulation theory in Section 5. Section 6 uses bisimulation to establish some non-trivial type isomorphisms and in Section 7 we prove a genericity property of bisimilarity.

2 Jump-With-Argument

Syntax Jump-With-Argument is a continuation-passing style calculus, extending the CPS calculus in [25]. Its types are given by

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

where I is any finite set. Binary sums are, of course, a special case of finite sums, $A_1 + A_2 \stackrel{\text{def}}{=} \sum_{i=1}^{2} A_i$, and the empty type, 0, is the empty sum. The type $\neg A$ is the type of functions (or continuations) that take an argument of type A and do not return.

$$\frac{}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} \mathtt{x}_i{:}A_i}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V_i{:}B_i \ \ (\forall i \in \{1,2\})}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} \langle V_1, V_2\rangle : B_1 \times B_2} \qquad \frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}B_1 \times B_2 \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}, \mathtt{y}_1{:}B_1, \mathtt{y}_2{:}B_2 \vdash^{\mathsf{n}} M}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{y}_1, \mathtt{y}_2\rangle.M}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash B_i \ \ (\forall i \in I) \quad \hat{\imath} \in I \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}B_{\hat{\imath}}}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} \langle \hat{\imath}, V\rangle : \textstyle\sum_{i\in I} B_i} \qquad \frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}\textstyle\sum_{i\in I} B_i \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}, \mathtt{y}_i{:}B_i \vdash^{\mathsf{n}} M_i \ \ (\forall i \in I)}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{y}_i\rangle.M_i\}_{i\in I}}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}, \mathtt{y}{:}B \vdash^{\mathsf{n}} M}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} \lambda\mathtt{y}.M : \neg B} \qquad \frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} U{:}\neg B \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}B}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} UV}$$

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

$$\frac{\overrightarrow{\mathtt{X}} \vdash C \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}B[C/\mathtt{Y}]}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} \langle C, V\rangle : \exists\mathtt{Y}.B} \qquad \frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}\exists\mathtt{Y}.B \quad \overrightarrow{\mathtt{X}}, \mathtt{Y}, \overrightarrow{\mathtt{x}{:}A}, \mathtt{y}{:}B \vdash^{\mathsf{n}} M}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.\ M}$$

Figure 1. Syntax of JWA, with type recursion

$$\begin{array}{lcl}
\mathtt{pm}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M & \rightsquigarrow & M[V/\mathtt{x}, V'/\mathtt{y}] \\
\mathtt{pm}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\ M_i\}_{i\in I} & \rightsquigarrow & M_{\hat{\imath}}[V/\mathtt{x}] \\
(\lambda\mathtt{x}.M)V & \rightsquigarrow & M[V/\mathtt{x}] \\
\mathtt{pm}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M & \rightsquigarrow & M[V/\mathtt{x}] \\
\mathtt{pm}\ \langle A, V\rangle\ \mathtt{as}\ \langle \mathtt{X}, \mathtt{x}\rangle.\ M & \rightsquigarrow & M[A/\mathtt{X}][V/\mathtt{x}]
\end{array}$$

Figure 2. C-machine transitions

$$\begin{array}{l}
\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M \\
\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\ M_i\}_{i\in I} \\
\mathtt{z}V \\
\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M \\
\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{X}, \mathtt{x}\rangle.\ M
\end{array}$$

Figure 3. Terminal commands

JWA has three judgements: *types* written $\overrightarrow{\mathtt{X}} \vdash A$, *values* written $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V{:}B$, and *nonreturning commands* written $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M$. The syntax is shown in Fig. 1. 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 $\times$, and we omit the rules for the types judgements $\overrightarrow{\mathtt{X}} \vdash A$, they just say that the free type identifiers in A are included in the set $\{\overrightarrow{\mathtt{X}}\}$.

Operational semantics To evaluate a command $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M$, simply apply the transitions (β-reductions) in Fig. 2 until a terminal command in Fig. 3 is reached. Every command M is either a redex or terminal (that is, terminals are β-normal forms). By determinism, either $M \rightsquigarrow^* T$ for unique terminal T, or else $M \rightsquigarrow^\omega$. This operational semantics is called the *C-machine*.

Examples JWA can be used as the target language for continuation-passing style transformations from direct-style programming language calculi, one such transform is the stack-passing transform from the Call-By-Push-Value to JWA [16]. We will not describe any such transforms in this paper. Instead we describe some examples of JWA programs. Familiarity with continuation-passing style programming is useful (see, e.g., [21]).

Example 1 *We define a polymorphic fixed point combinator* Fix *as* $\mathtt{Fix} \stackrel{\text{def}}{=} \Phi(\mathtt{fold}\ \mathtt{V})$*, where*

$$\mathtt{V} \stackrel{\text{def}}{=} \lambda\langle \mathtt{w}, \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}\rangle\rangle\rangle.\mathtt{f}\langle \mathtt{x}, \lambda\mathtt{y}.\Phi(\mathtt{w})\langle \mathtt{X}, \langle \mathtt{y}, \mathtt{f}\rangle\rangle\rangle\rangle$$
$$\Phi(W) \stackrel{\text{def}}{=} \lambda\mathtt{u}.\mathtt{pm}\ W\ \mathtt{as}\ \mathtt{fold}\ \mathtt{v}.\mathtt{v}\langle \mathtt{fold}\ \mathtt{v}, \mathtt{u}\rangle$$

and notation $\lambda\langle \mathtt{w}, \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}\rangle\rangle\rangle.M$ *is short for*

$$\lambda\mathtt{a}.\mathtt{pm}\ \mathtt{a}\ \mathtt{as}\ \langle \mathtt{w}, \mathtt{u}\rangle.\mathtt{pm}\ \mathtt{u}\ \mathtt{as}\ \langle \mathtt{X}, \mathtt{b}\rangle.\mathtt{pm}\ \mathtt{b}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{f}\rangle.M$$

(we use this kind of short hand in examples throughout).

We assign type $\neg\exists\mathtt{X}.\,\mathtt{X} \times \neg(\mathtt{X} \times \neg\mathtt{X})$ *to* Fix *by assigning the recursive type* $\mu\mathtt{Z}.\neg(\mathtt{Z} \times \exists\mathtt{X}.\,\mathtt{X} \times \neg(\mathtt{X} \times \neg\mathtt{X}))$ *to* fold V.

The calculation

$$\begin{array}{ll} & \mathtt{Fix}\langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}\rangle\rangle \\ \leadsto & \mathtt{pm\ fold\ V\ as\ fold\ v.v}\langle \mathtt{fold\ v}, \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}\rangle\rangle\rangle \\ \leadsto & \mathtt{V}\langle \mathtt{fold\ V}, \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}\rangle\rangle\rangle \\ \leadsto^4 & \mathtt{f}\langle \mathtt{x}, \lambda \mathtt{y}.\Phi(\mathtt{fold\ V})\langle \mathtt{X}, \langle \mathtt{y}, \mathtt{f}\rangle\rangle\rangle \\ = & \mathtt{f}\langle \mathtt{x}, \lambda \mathtt{y}.\mathtt{Fix}\langle \mathtt{X}, \langle \mathtt{y}, \mathtt{f}\rangle\rangle\rangle \end{array}$$

shows that Fix *is a solution to the fixed point equation*

$$\vdash^{\mathtt{v}} \mathtt{Fix} =_\beta \lambda\langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}\rangle\rangle.\mathtt{f}\langle \mathtt{x}, \lambda \mathtt{y}.\mathtt{Fix}\langle \mathtt{X}, \langle \mathtt{y}, \mathtt{f}\rangle\rangle\rangle : \neg\exists \mathtt{X}.\, \mathtt{X} \times \neg(\mathtt{X} \times \neg \mathtt{X})$$

We encode finite lists with elements of type X as type $\mathtt{List(X)} \stackrel{\text{def}}{=} \mu \mathtt{Z}.\, 1 + \mathtt{X} \times \mathtt{Z}$ and let $\mathtt{nil} \stackrel{\text{def}}{=} \mathtt{fold}\ \langle 1, \langle\rangle\rangle$, $\mathtt{cons}\langle U, V\rangle \stackrel{\text{def}}{=} \mathtt{fold}\ \langle 2, \langle U, V\rangle\rangle$, and $[U_1, ..., U_n] \stackrel{\text{def}}{=} \mathtt{cons}\langle U_1, ..., \mathtt{cons}\langle U_n, \mathtt{nil}\rangle ...\rangle$. We use the syntactic sugar

$$\begin{array}{l} \mathtt{pm}\ V\ \mathtt{as}\ \{\mathtt{nil}.M, \mathtt{cons}\langle \mathtt{h}, \mathtt{t}\rangle.N\} \stackrel{\text{def}}{=} \\ \quad \mathtt{pm}\ V\ \mathtt{as\ fold\ x.\,pm\ x\ as}\ \{\langle 1, \langle\rangle\rangle.M, \langle 2, \langle \mathtt{h}, \mathtt{t}\rangle\rangle.N\} \end{array}$$

where x is fresh.

Example 2 *Using the fixed point combinator* Fix *we can write a polymorphic, recursive list reversal function*

$$\mathtt{reverse} : \neg\exists \mathtt{X}.\, \mathtt{List(X)} \times \neg \mathtt{List(X)}$$

that takes a type X*, a list* u *of type* List(X)*, and a continution* k *of type* $\neg$List(X) *and applies* k *to the reversal of the list* u*.*

$$\begin{array}{l} \mathtt{reverse} \stackrel{\text{def}}{=} \lambda\langle \mathtt{X}, \langle \mathtt{u}, \mathtt{k}\rangle\rangle.\, \mathtt{Fix}\langle \mathtt{List(X)} \times \mathtt{List(X)}, \\ \qquad\qquad \langle\langle \mathtt{u}, \mathtt{nil}\rangle, \mathtt{reverse}'\rangle\rangle \\ \mathtt{reverse}' \stackrel{\text{def}}{=} \lambda\langle\langle \mathtt{u}, \mathtt{v}\rangle, \mathtt{r}\rangle. \\ \qquad \mathtt{pm\ u\ as}\ \{\mathtt{nil}.\,\mathtt{k\,v}, \\ \qquad\qquad \mathtt{cons}\langle \mathtt{x}, \mathtt{w}\rangle.\,\mathtt{r}\langle \mathtt{w}, \mathtt{cons}\langle \mathtt{x}, \mathtt{v}\rangle\rangle\}\rangle \end{array}$$

In the next example we use Boolean and natural number types and the notation:

$$\begin{array}{ll} \mathtt{Bool} \stackrel{\text{def}}{=} 1 + 1 & \mathtt{Nat} \stackrel{\text{def}}{=} \mu \mathtt{X}.1 + \mathtt{X} \\ \mathtt{false} \stackrel{\text{def}}{=} \langle 1, \langle\rangle\rangle & \mathtt{zero} \stackrel{\text{def}}{=} \mathtt{fold}\ \langle 1, \langle\rangle\rangle \\ \mathtt{true} \stackrel{\text{def}}{=} \langle 2, \langle\rangle\rangle & \mathtt{S}\ V \stackrel{\text{def}}{=} \mathtt{fold}\ \langle 2, V\rangle \end{array}$$

We use syntactic sugar for pattern matching, like we did for lists, and write the negation function neg on Booleans as:

$$\mathtt{neg} \stackrel{\text{def}}{=} \lambda\langle \mathtt{b}, \mathtt{k}\rangle.\mathtt{pm\ b\ as}\ \{\mathtt{false.\,k\,true}, \mathtt{true.\,k\,false}\}$$

and the parity function even on natural numbers as:

$$\begin{array}{l} \mathtt{even} \stackrel{\text{def}}{=} \lambda \mathtt{p}.\, \mathtt{Fix}\langle \mathtt{Nat} \times \neg \mathtt{Bool}, \langle \mathtt{p}, \mathtt{even}'\rangle\rangle \\ \mathtt{even}' \stackrel{\text{def}}{=} \lambda\langle\langle \mathtt{x}, \mathtt{k}\rangle, \mathtt{e}\rangle.\, \mathtt{pm\ x\ as}\ \{ \\ \qquad\qquad \mathtt{zero.\,k\,true}, \\ \qquad\qquad \mathtt{S\,y.\,pm\ y\ as}\ \{ \\ \qquad\qquad\qquad \mathtt{zero.\,k\,false}, \\ \qquad\qquad\qquad \mathtt{S\,z.\,e}\langle \mathtt{z}, \mathtt{k}\rangle\}\} \end{array}$$

Example 3 *Consider the abstract data type*

$$\mathtt{Semaphore} \stackrel{\text{def}}{=} \exists \mathtt{X}.\mathtt{X} \times \neg(\mathtt{X} \times \neg \mathtt{X}) \times \neg(\mathtt{X} \times \neg \mathtt{Bool})$$

The following values are two equivalent implementations where the three components of the triple are (1) an initial value, (2) a function that "flips" a value, and (3) a third component that reads the parity of the number of flips.

$$\begin{array}{l} \mathtt{semaphore1} \stackrel{\text{def}}{=} \langle \mathtt{Bool}, \langle \mathtt{true}, \mathtt{neg}, \mathtt{id}\rangle\rangle \\ \mathtt{semaphore2} \stackrel{\text{def}}{=} \langle \mathtt{Nat}, \langle \mathtt{zero}, \mathtt{succ}, \mathtt{even}\rangle\rangle \end{array}$$

where $\mathtt{id} \stackrel{\text{def}}{=} \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\,\mathtt{k\,x}$ *and* $\mathtt{succ} \stackrel{\text{def}}{=} \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\,\mathtt{k}(\mathtt{S\,x})$.

3 Ultimate Patterns

We are going to define a LTS describing the interaction between a program and its environment. In each labelled transition, the program passes a value V to the environment, or vice versa. Some parts of V are visible to the recipient. Other parts are hidden to the recipient, who regards them as free identifiers. "Ultimate pattern matching" is the process of decomposing V into an "ultimate value pattern" (the visible part) and "filling" (the hidden part). It is essential to our theory that any value V has a unique decomposition.

In the absence of polymorphism, this distinction is quite straightforward [14]: V consists of its first-order structure, which is visible, and functions, which are hidden. In our polymorphic setting, V will also contain types, which are hidden. More subtly, V will contain *opaque values*, i.e., values whose type has been passed during the interaction. These are of two kinds. In the case of an output from the program to the environment:

1. An *output opaque* value, one whose type was sent by the program either (a) in an earlier transition, or (b) in the current transition (meaning that the type appears in V). Such values are hidden to the environment.

2. An *input opaque* value, one whose type was previously received from the environment. Such a value must itself have been previously received by the program, who regards it as a free identifier. This identifier is deemed visible to the program. (The rationale for this is that the environment could have chosen to send the type Nat and a different number for each opaque value of that type.)

Thus the ultimate value pattern of V consists of the first-order structure and input opaque values. The filling consists of the functions, types and output opaque values. Syntactically, an ultimate value pattern is a value with free type and value identifiers and a filling is a substitution of types and values for the free identifiers.

The "signature" of the filling is described by a three-part type context of the form

$$\Delta \;=\; \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A},$$

which we call a *flat type context*; we use the meta-variables Ξ, Υ to range over type identifiers; $\overrightarrow{\Xi}$, $\overrightarrow{\Upsilon}$ are sequences of type identifiers, not necessarily distinct; and $\overrightarrow{\mathtt{X}} \vdash \overrightarrow{\Xi}$ means that the type identifiers in $\overrightarrow{\Xi}$ are drawn from $\overrightarrow{\mathtt{X}}$. We write $\emptyset$ for the empty type context.

We use notation $\Delta^\dagger$ to refer to the first part of Δ (the type identifiers) and $\Delta^\ddagger$ to the first two parts (the type identifiers and opaque value identifiers), that is, $\Delta^\dagger = \overrightarrow{\mathtt{X}}$ and $\Delta^\ddagger = \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}$, if $\Delta = \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A}$. We write $\mathrm{dom}(\Delta)$ for the set $\{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}}, \overrightarrow{\mathtt{f}}\}$.

When Δ and Θ are two flat type contexts,

$$\Delta = \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A}, \quad \Theta = \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{y}{:}\Upsilon}, \overrightarrow{\mathtt{g}{:}\neg B},$$

we write Δ, Θ for the combined flat type context

$$\Delta, \Theta = \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{y}{:}\Upsilon}, \overrightarrow{\mathtt{f}{:}\neg A}, \overrightarrow{\mathtt{g}{:}\neg B},$$

when the type identifiers $\overrightarrow{\mathtt{Y}} = \Theta^\dagger$ do not occur in Δ.

The ultimate value patterns p are given by a judgement

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash p : D \tag{1}$$

where Δ has the form $\Delta = \overrightarrow{\mathtt{Z}}, \overrightarrow{\mathtt{y}{:}\Upsilon}, \overrightarrow{\mathtt{f}{:}\neg A}$ and the types satisfy $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{Y}} \vdash D$; $\overrightarrow{\mathtt{X}} \vdash \overrightarrow{\Xi}$; $\overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{Z}} \vdash \overrightarrow{\Upsilon}$; $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{Z}} \vdash \overrightarrow{A}$. These constraints on the types and type identifiers are implicit in the rules defining the judgement in Fig. 4.

Its informal meaning is: Suppose that the current sender has previously received types and input opaque values known to him as $\overrightarrow{\mathtt{X}}$ and $\overrightarrow{\mathtt{x}}$ respectively, and has previously sent types known to the (then and current) recipient as $\overrightarrow{\mathtt{Y}}$. Then p is an ultimate value pattern for a value of type D, containing types, output opaque values and functions known henceforth to the recipient by the identifiers in Δ.

Syntactically, an ultimate value pattern is a value

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{Y}}, \Delta \vdash^{\mathtt{v}} p : D.$$

Example 4 *If* $\mathtt{X}$ *is an "input type" and we have, say,* m *opaque input values* $\mathtt{x}_1, ..., \mathtt{x}_m$ *of type* $\mathtt{X}$*, the ultimate value patterns of type* $\mathtt{List(X)}$ *all have the form:*

$$\mathtt{X}, \mathtt{x}_1{:}\mathtt{X}, ..., \mathtt{x}_m{:}\mathtt{X} \parallel \emptyset \mid \emptyset \vdash [\mathtt{x}_{i_1}, ..., \mathtt{x}_{i_n}] : \mathtt{List(X)}$$

where $i_1, ..., i_n \in \{1, ..., m\}$*. That is, the values of type* $\mathtt{List(X)}$ *we can form in the type context* $\mathtt{X}, \mathtt{x}_1{:}\mathtt{X}, ..., \mathtt{x}_m{:}\mathtt{X}$ *are arbitrary length lists with input opaque values drawn from* $\mathtt{x}_1, ..., \mathtt{x}_m$ *as elements.*

$$\frac{(\mathtt{x}{:}\mathtt{X}) \in \overrightarrow{\mathtt{x}{:}\Xi}}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \emptyset \vdash \mathtt{x} : \mathtt{X}}$$

$$\frac{\overrightarrow{\mathtt{Y}} \vdash \Upsilon}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \mathtt{y}{:}\Upsilon \vdash \mathtt{y} : \Upsilon}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta_j \vdash p_j : A_j \;\; (\forall j \in \{1,2\})}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta_1, \Delta_2 \vdash \langle p_1, p_2 \rangle : A_1 \times A_2}$$

$$\frac{\hat{\imath} \in I \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash p : A_{\hat{\imath}}}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash \langle \hat{\imath}, p \rangle : \sum_{i \in I} A_i}$$

$$\frac{}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \mathtt{f}{:}\neg A \vdash \mathtt{f} : \neg A}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash p : A[\mu\mathtt{X}.A/\mathtt{X}]}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash \mathtt{fold}\, p : \mu\mathtt{X}.A}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}}, \mathtt{X} \mid \Delta \vdash p : A}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \mathtt{X}, \Delta \vdash \langle \mathtt{X}, p \rangle : \exists\mathtt{X}.A}$$

Figure 4. Ultimate Value Patterns

Example 5 *If* $\mathtt{X}$ *is an "output type", the ultimate value patterns of type* $\mathtt{List(X)}$ *are lists with (distinct, fresh) output opaque values as elements:*

$$\emptyset \parallel \mathtt{X} \mid \overrightarrow{\mathtt{x}}{:}\mathtt{X} \vdash [\overrightarrow{\mathtt{x}}] : \mathtt{List(X)}.$$

Example 6 *The ultimate value patterns of the argument type to the* `reverse` *function, Example 2, have the form:*

$$\emptyset \parallel \emptyset \mid \mathtt{X}, \overrightarrow{\mathtt{x}}{:}\mathtt{X}, \mathtt{f}{:}\neg\mathtt{List(X)} \vdash \langle \mathtt{X}, \langle [\overrightarrow{\mathtt{x}}], \mathtt{f} \rangle \rangle : \exists\mathtt{X}.\, \mathtt{List(X)} \times \neg\mathtt{List(X)}.$$

Corresponding to the three-part flat type contexts, we introduce three-part *flat substitutions*,

$$u \;=\; \overrightarrow{B/\mathtt{X}}; \overrightarrow{U/\mathtt{x}}; \overrightarrow{F/\mathtt{f}}.$$

We use the meta-variables F, G to range over function values, viz., λ-values and identifiers of function type. We write ϵ for the empty substitution. We define $t[u] = t[\overrightarrow{B/\mathtt{X}}][\overrightarrow{U/\mathtt{x}}, \overrightarrow{F/\mathtt{f}}]$ if t is a term, and $\Theta[u] = \Theta[\overrightarrow{B/\mathtt{X}}]$ if Θ is a type or a type context, and $\mathrm{dom}(u) = \{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}}, \overrightarrow{\mathtt{f}}\}$.

A filling of the ultimate value pattern (1) is a flat substitution u with "signature" Δ. (We postpone the formal typing judgement for flat substitutions till Sect. 4.)

Theorem 1 (ultimate value pattern matching) *Any value* $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A[B/\mathtt{Y}]} \vdash^{\mathsf{v}} U : D[\overrightarrow{B/\mathtt{Y}}]$*, where* $\overrightarrow{\mathtt{X}} \vdash \overrightarrow{B}$*, uniquely decomposes* $U = p[w]$ *into an ultimate value pattern* $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash p : D$ *and flat substitution* w *with* $dom(w) = dom(\Delta)$*. That is,* p*,* w*, and* Δ *are unique up to the choice of identifiers in* $dom(\Delta)$*.*

In particular, any closed value U of closed type D uniquely decomposes $U = p[w]$ into an ultimate value pattern $\emptyset \parallel \emptyset \mid \Delta \vdash p : D$ and flat substitution w.

Example 7 *The semaphores from Example 3 decompose*

$$\begin{aligned}\mathtt{semaphore1} &= \langle \mathtt{Bool}, \langle \mathtt{true}, \mathtt{neg}, \mathtt{id}\rangle\rangle \\ &= \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}, \mathtt{g}\rangle\rangle[\mathtt{Bool}/\mathtt{X}; \mathtt{true}/\mathtt{x}; \mathtt{neg}/\mathtt{f}; \mathtt{idg}],\\ \mathtt{semaphore2} &= \langle \mathtt{Nat}, \langle \mathtt{zero}, \mathtt{succ}, \mathtt{even}\rangle\rangle \\ &= \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}, \mathtt{g}\rangle\rangle[\mathtt{Nat}/\mathtt{X}; \mathtt{zero}/\mathtt{x}; \mathtt{succ}/\mathtt{f}, \mathtt{even}/\mathtt{g}],\end{aligned}$$

where the ultimate value pattern $\langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}, \mathtt{g}\rangle\rangle$ *has type*

$$\emptyset \parallel \emptyset \mid \mathtt{X}, \mathtt{x}{:}\mathtt{X}, \mathtt{f}{:}\neg(\mathtt{X}\times\neg\mathtt{X}), \mathtt{g}{:}\neg(\mathtt{X}\times\neg\mathtt{Nat}) \vdash \langle \mathtt{X}, \langle \mathtt{x}, \mathtt{f}, \mathtt{g}\rangle\rangle : \mathtt{Semaphore}$$

and $\mathtt{Semaphore} = \exists \mathtt{X}.\mathtt{X}\times\neg(\mathtt{X}\times\neg\mathtt{X})\times\neg(\mathtt{X}\times\neg\mathtt{Bool})$.

The following technical lemma is used in the substitution proofs in Sect. 5.

Lemma 2 *If* $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{Z}}, \overrightarrow{\mathtt{z}{:}\Upsilon} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash p : D$ *and* $\overrightarrow{\mathtt{Z}} \vdash \overrightarrow{\Upsilon}$*, there are unique* $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi} \parallel \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{Z}} \mid \overrightarrow{\mathtt{z}'{:}\Upsilon'}, \Delta \vdash p' : D$ *and renaming* r *from* $\overrightarrow{\mathtt{z}'{:}\Upsilon'}$ *to* $\overrightarrow{\mathtt{z}{:}\Upsilon}$ *such that* $p = p'[r]$*.*

4 LTS

We now use flat type contexts, flat substitutions, and ultimate patterns to define our LTS.

The judgement $\Delta_{\mathsf{i}} \dashv\vdash \Delta_{\mathsf{o}}$ means that the flat type contexts Δ_{i} and Δ_{o} form a joint *input-output context*,

$$\frac{\overrightarrow{\mathtt{X}} \vdash \overrightarrow{\Xi} \quad \overrightarrow{\mathtt{Y}} \vdash \overrightarrow{\Upsilon} \quad \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{Y}} \vdash \overrightarrow{A}, \overrightarrow{B}}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A} \dashv\vdash \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{y}{:}\Upsilon}, \overrightarrow{\mathtt{g}{:}\neg B}}$$

The judgement $\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u$ means that the flat substitution u has the "signature" (output context) Δ_{o} in the input context Δ_{i},

$$\frac{\begin{array}{c}\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A} \dashv\vdash \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{y}{:}\Upsilon}, \overrightarrow{\mathtt{g}{:}\neg B} \\ \overrightarrow{\mathtt{X}} \vdash \overrightarrow{C} \\ \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A[\overrightarrow{C/\mathtt{Y}}]} \vdash^{\mathsf{v}} \overrightarrow{V : \Upsilon[\overrightarrow{C/\mathtt{Y}}]}, \overrightarrow{G : \neg B[\overrightarrow{C/\mathtt{Y}}]}\end{array}}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \overrightarrow{\mathtt{f}{:}\neg A} \parallel \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{y}{:}\Upsilon}, \overrightarrow{\mathtt{g}{:}\neg B} \vdash \overrightarrow{C/\mathtt{Y}}; \overrightarrow{V/\mathtt{y}}; \overrightarrow{G/\mathtt{g}}}$$

We will write $\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash p : A$ to mean $\Delta_{\mathsf{i}} \dashv\vdash \Delta_{\mathsf{o}}$ and $\Delta_{\mathsf{i}} \dashv\vdash \Delta_{\mathsf{o}}, \Delta$ and $\Delta_{\mathsf{i}}^{\ddagger} \parallel \Delta_{\mathsf{o}}^{\dagger} \mid \Delta \vdash p : A$.

An *ultimate terminal command pattern* $a = \mathtt{f}\, p$ is an input function identifier $\mathtt{f}$ applied to an ultimate value pattern p, defined by the judgement $\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash a$,

$$\frac{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash p : A \quad (\mathtt{f}{:}\neg A) \in \Delta_{\mathsf{i}}}{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash \mathtt{f}\, p}$$

Theorem 3 (ultimate command pattern matching) *If* $\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u$*, any terminal command* $\Delta_{\mathsf{i}}[u] \vdash^{\mathsf{n}} T$ *uniquely decomposes* $T = a[w]$ *into an ultimate terminal command pattern* a *and flat substitution* w *such that*

$$\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash a, \quad \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}}, \Delta \vdash u, w.$$

There are two kinds of LTS *states*, active and passive. A passive state u is just a flat substitution. An active state $c = u; M$ is a flat substitution plus a command component M,

$$\frac{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u \quad \Delta_{\mathsf{i}}[u] \vdash^{\mathsf{n}} M}{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u; M}$$

A *label* is an ultimate terminal command pattern a. Relative to an input-output context $\Delta_{\mathsf{i}} \| \Delta_{\mathsf{o}}$, we call a an *output label* if $\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash a$ and an *input label* if $\Delta_{\mathsf{o}} \parallel \Delta_{\mathsf{i}} \mid \Delta \vdash a$.

There are three kinds of transitions:

A *silent transition* is from an active state, with a non-terminal command, to another active state. It corresponds to a C-machine transition on the command component.

$$\frac{M \rightsquigarrow N}{u; M \rightsquigarrow u; N}$$

A *labelled output transition* is from an active state with terminal command component to a passive state.

$$\frac{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash a \quad \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u \quad \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}}, \Delta \vdash u, w}{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash u; a[w] \xrightarrow{a} u, w}$$

Sometimes we write "u, w" as "$u|w$" on the right of the output transition arrow to indicate the portion w that is the filling of the ultimate command pattern in the label.

A *labelled input transition* is from a passive state to an active state.

$$\frac{\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u \quad \Delta_{\mathsf{o}} \parallel \Delta_{\mathsf{i}} \mid \Delta \vdash a}{\Delta_{\mathsf{i}} \mid \Delta \parallel \Delta_{\mathsf{o}} \vdash u \xrightarrow{a} u; a[u]}$$

5 Bisimulation

A bisimulation is a binary relation between states in the same input-ouput context. Since there are two kinds of states, active and passive, there are two classes of relations,

$$\mathcal{R} \subseteq \{(\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash c, c')\},\ \mathcal{S} \subseteq \{(\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u, u')\}.$$

(We sometimes write $(\Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \vdash x, x') \in \mathcal{X}$ with infix notation $\Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \vdash x \;\mathcal{X}\; x$, where $\mathcal{X}$ is a passive or active relation and x, x' are passive or active states.)

Each class is a complete lattice ordered by subset inclusion. The operators

$$\begin{aligned}
\mathcal{S}^\sharp \stackrel{\text{def}}{=} \{&(\Delta_\mathsf{i}, \Delta \parallel \Delta_\mathsf{o} \vdash c, c') \mid \\
&\exists u, u', a.\ \Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \vdash u\ \mathcal{S}\ u'\ \& \\
&\qquad \Delta_\mathsf{i} \mid \Delta \parallel \Delta_\mathsf{o} \vdash u \xrightarrow{a} c\ \& \\
&\qquad \Delta_\mathsf{i} \mid \Delta \parallel \Delta_\mathsf{o} \vdash u' \xrightarrow{a} c'\} \\
\mathcal{R}^\flat \stackrel{\text{def}}{=} \{&(\Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \vdash u, u') \mid \\
&\forall \Delta, a, c, c'.\ (\Delta_\mathsf{i} \mid \Delta \parallel \Delta_\mathsf{o} \vdash u \xrightarrow{a} c\ \& \\
&\qquad \Delta_\mathsf{i} \mid \Delta \parallel \Delta_\mathsf{o} \vdash u' \xrightarrow{a} c') \\
&\qquad \Rightarrow \Delta_\mathsf{i}, \Delta \parallel \Delta_\mathsf{o} \vdash c\ \mathcal{R}\ c'\}
\end{aligned}$$

form a galois connection,

$$\mathcal{S}^\sharp \subseteq \mathcal{R} \text{ iff } \mathcal{S} \subseteq \mathcal{R}^\flat. \tag{2}$$

We define a bisimulation operator $\mathcal{B}$ as the map from passive to active relations:

$$\begin{aligned}
\mathcal{B}(\mathcal{S}) \stackrel{\text{def}}{=} \{&(\Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \vdash c, c') \mid \\
&(c \rightsquigarrow^\omega\ \ \&\ c' \rightsquigarrow^\omega) \vee \\
&\exists \Delta, a, u, w, u', w'. \\
&\quad \Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \mid \Delta \vdash c \rightsquigarrow^* \xrightarrow{a} u \mid w\ \& \\
&\quad \Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \mid \Delta \vdash c' \rightsquigarrow^* \xrightarrow{a} u' \mid w'\ \& \\
&\quad \Delta_\mathsf{i} \parallel \Delta_\mathsf{o}, \Delta \vdash u, w\ \ \mathcal{S}\ \ u', w'\}
\end{aligned}$$

where $\Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \mid \Delta \vdash c \rightsquigarrow^* \xrightarrow{a} u \mid w$ means

$$\exists d.\ c \rightsquigarrow^* d\ \&\ \Delta_\mathsf{i} \parallel \Delta_\mathsf{o} \mid \Delta \vdash d \xrightarrow{a} u \mid w.$$

We say that a relation $\mathcal{R}$ is a *bisimulation* if $\mathcal{R} \subseteq \mathcal{B}(\mathcal{R}^\flat)$. The relational operators are all monotone and therefore there exists a greatest bisimulation, which we denote $\approx$. It is the greatest fixed point $\approx\ = \mathcal{B}(\approx^\flat)$.

It is sometimes more convenient to work with passive relations. A *passive bisimulation* is a relation $\mathcal{S} \subseteq \mathcal{B}(\mathcal{S})^\flat$. By the galois connection (2), $\mathcal{S} \subseteq \mathcal{B}(\mathcal{S})^\flat$ iff $\mathcal{S}^\sharp \subseteq \mathcal{B}(\mathcal{S})$. Moreover, $\mathcal{S}^\sharp$ is a bisimulation if $\mathcal{S}$ is a passive bisimulation, $\mathcal{R}^\flat$ is a passive bisimulation if $\mathcal{R}$ is a bisimulation, and $\approx^\flat$ is the greatest passive bisimulation.

Example 8 *Recall from Example 7 the flat type context*

$$\Delta = \mathtt{X}, \mathtt{x}{:}\mathtt{X}, \mathtt{f}{:}\neg(\mathtt{X}\times\neg\mathtt{X}), \mathtt{g}{:}\neg(\mathtt{X}\times\neg\mathtt{Bool})$$

and two flat substitutions $\emptyset \parallel \Delta \vdash v_1, v_2$*,*

$$\begin{aligned}
v_1 &= \mathtt{Bool}/\mathtt{X}, \mathtt{true}/\mathtt{x}, \mathtt{neg}/\mathtt{f}, \mathtt{id}/\mathtt{g}, \\
v_2 &= \mathtt{Nat}/\mathtt{X}, \mathtt{zero}/\mathtt{x}, \mathtt{succ}/\mathtt{f}, \mathtt{even}/\mathtt{g}.
\end{aligned}$$

They are related by the passive bisimulation

$$\begin{aligned}
\mathcal{S} = \{&(\Delta_\mathsf{i} \parallel \Delta_{\vec{n}} \vdash u_{\vec{n}}, u'_{\vec{n}}) \mid \vec{n} \in \mathbb{N}^*\ \& \\
&\quad \Delta_\mathsf{i} \dashv\vdash \mathtt{X}\ \&\ \Delta_{\vec{n}} = \Delta, \mathtt{x}_1{:}\mathtt{X}, \ldots, \overrightarrow{\mathtt{x}_{|\vec{n}|}{:}\mathtt{X}}\ \& \\
&\quad u_{\vec{n}} = v_1, \overrightarrow{\beta(n)/\mathtt{x}}\ \&\ u'_{\vec{n}} = v_2, \overrightarrow{\nu(n)/\mathtt{x}}\}
\end{aligned}$$

where $\nu(n) \stackrel{\text{def}}{=} \mathtt{S}^n\mathtt{zero}$*,* $\beta(n) \stackrel{\text{def}}{=} \begin{cases} \mathtt{true} & \text{if } n \text{ is even} \\ \mathtt{false} & \text{if } n \text{ is odd.} \end{cases}$

To see that $\mathcal{S}$ *is a passive bisimulation first observe that the only input labels from the passive states related by* $\mathcal{S}$ *are* $a_i = \mathtt{f}\langle \mathtt{x}_i, \mathtt{k}\rangle$ *and* $b_i = \mathtt{g}\langle \mathtt{x}_i, \mathtt{k}\rangle$*, so*

$$\begin{aligned}
\mathcal{S}^\sharp = \{&(\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{X} \parallel \Delta_{\vec{n}} \vdash u_{\vec{n}}; a_i[u_{\vec{n}}], u'_{\vec{n}}; a_i[u'_{\vec{n}}]), \\
&(\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{Bool} \parallel \Delta_{\vec{n}} \vdash u_{\vec{n}}; b_i[u_{\vec{n}}], u'_{\vec{n}}; b_i[u'_{\vec{n}}])\}.
\end{aligned}$$

We calculate

$$a_i[u_{\vec{n}}] = \mathtt{neg}\langle \beta(n_i), \mathtt{k}\rangle \rightsquigarrow^* \mathtt{k}\,\beta(n_i{+}1)$$

$$\begin{aligned}
&\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{X} \parallel \Delta_{\vec{n}} \mid \mathtt{x}_{|\vec{n}|+1}{:}\mathtt{X} \vdash \\
&\quad u_{\vec{n}}; a_i[u_{\vec{n}}] \rightsquigarrow^* \xrightarrow{\mathtt{k}\,\mathtt{y}} u_{\vec{n}} \mid \beta(n_i{+}1)/\mathtt{x}_{|\vec{n}|+1}
\end{aligned}$$

$$a_i[u'_{\vec{n}}] = \mathtt{succ}\langle \nu(n_i), \mathtt{k}\rangle \rightsquigarrow^* \mathtt{k}\,\nu(n_i{+}1)$$

$$\begin{aligned}
&\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{X} \parallel \Delta_{\vec{n}} \mid \mathtt{x}_{|\vec{n}|+1}{:}\mathtt{X} \vdash \\
&\quad u'_{\vec{n}}; a_i[u'_{\vec{n}}] \rightsquigarrow^* \xrightarrow{\mathtt{k}\,\mathtt{y}} u'_{\vec{n}} \mid \nu(n_i{+}1)/\mathtt{x}_{|\vec{n}|+1}
\end{aligned}$$

$$\begin{aligned}
\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{X} \parallel \Delta_{\vec{n}}, \mathtt{x}_{|\vec{n}|+1}{:}\mathtt{X} \vdash u_{\vec{n}}, \beta(n_i{+}1)/\mathtt{x}_{|\vec{n}|+1} &= \\
u_{\vec{n},(n_i+1)}\ \mathcal{S}\ u'_{\vec{n},(n_i+1)} &= \\
u'_{\vec{n}}, \nu(n_i{+}1)/\mathtt{x}_{|\vec{n}|+1} &
\end{aligned}$$

and

$$b_i[u_{\vec{n}}] = \mathtt{id}\langle \beta(n_i), \mathtt{k}\rangle \rightsquigarrow^* \mathtt{k}\,\beta(n_i)$$

$$\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{Bool} \parallel \Delta_{\vec{n}} \mid \emptyset \vdash u_{\vec{n}}; b_i[u_{\vec{n}}] \rightsquigarrow^* \xrightarrow{\mathtt{k}\,\beta(n_i)} u_{\vec{n}} \mid \epsilon$$

$$b_i[u'_{\vec{n}}] = \mathtt{even}\langle \nu(n_i), \mathtt{k}\rangle \rightsquigarrow^* \mathtt{k}\,\beta(n_i)$$

$$\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{Bool} \parallel \Delta_{\vec{n}} \mid \emptyset \vdash u'_{\vec{n}}; b_i[u'_{\vec{n}}] \rightsquigarrow^* \xrightarrow{\mathtt{k}\,\beta(n_i)} u'_{\vec{n}} \mid \epsilon$$

$$\Delta_\mathsf{i}, \mathtt{k}{:}\neg\mathtt{Bool} \parallel \Delta_{\vec{n}} \vdash u_{\vec{n}}\ \mathcal{S}\ u'_{\vec{n}}.$$

We conclude that $\mathcal{S}^\sharp \subseteq \mathcal{B}(\mathcal{S})$*, so* $\mathcal{S}$ *is a passive bisimulation and* $\emptyset \parallel \Delta \vdash v_1 \approx^\flat v_2$*.*

It is easy to see that $\approx$ and $\approx^\flat$ are equivalence relations (reflexive, transitive, and symmetric) and are preserved under renaming (of input context identifiers and type identifiers). Moreover, they are preserved under substitution:

Lemma 4 (preservation under substitution)
If $\Delta \parallel \Theta \vdash v \approx^\flat v'$ *and* $\Delta \dashv\vdash \Delta_\mathsf{o}$*,*

1. $\Delta, \Theta \parallel \Delta_\mathsf{o} \vdash c \approx c'$ *implies* $\Delta \parallel \Delta_\mathsf{o} \vdash c[v] \approx c'[v']$*, and*
2. $\Delta, \Theta \parallel \Delta_\mathsf{o} \vdash u \approx^\flat u'$ *implies* $\Delta \parallel \Delta_\mathsf{o} \vdash u[v] \approx^\flat u'[v']$.

Proof outline We have proved this result by generalizing the "alternating tables" from the proof for the monomorphic calculus [14] to type identifiers, open types, and type substitutions. Nonetheless, here we outline the argument in a notationally simpler "relational" formulation, akin to the substitutivity proofs in [13, 23].

We define the relation substitution operation $\mathcal{X}[\mathcal{S}]$,

$$\frac{\begin{array}{c}\Delta, \Theta, \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash x \ \mathcal{X} \ x' \\ \Delta, \Theta_{\mathsf{i}} \parallel \Theta, \Theta_{\mathsf{o}} \vdash v, w \ \mathcal{S} \ v', w' \\ \Delta, \Theta_{\mathsf{i}}, \Delta_{\mathsf{i}} \dashv\vdash \Theta_{\mathsf{o}}, \Delta_{\mathsf{o}}\end{array}}{\Delta, \Theta_{\mathsf{i}}, \Delta_{\mathsf{i}} \parallel \Theta_{\mathsf{o}}, \Delta_{\mathsf{o}} \vdash w, x[v] \ \ \mathcal{X}[\mathcal{S}] \ \ w', x'[v']}$$

where $\mathcal{S}$ is a passive relation and $\mathcal{X}$ is an active or passive relation.

The Lemma follows from the more general result

$$\precsim[\precsim^{\flat}] \subseteq \precsim \quad \text{and} \quad \precsim^{\flat}[\precsim^{\flat}] \subseteq \precsim^{\flat}.$$

We prove this co-inductively by exhibiting active and passive bisimulations $\mathcal{R}_\omega \supseteq \precsim[\precsim^{\flat}]$ and $\mathcal{S}_\omega \supseteq \precsim^{\flat}[\precsim^{\flat}]$, namely

$$\begin{array}{lll} \mathcal{R}_\omega \stackrel{\text{def}}{=} \bigcup_{m<\omega} \mathcal{R}_m, & \mathcal{R}_0 \stackrel{\text{def}}{=} \precsim, & \mathcal{R}_{m+1} \stackrel{\text{def}}{=} \mathcal{R}_m[\precsim^{\flat}], \\ \mathcal{S}_\omega \stackrel{\text{def}}{=} \bigcup_{m<\omega} \mathcal{S}_m, & \mathcal{S}_0 \stackrel{\text{def}}{=} \precsim^{\flat}, & \mathcal{S}_{m+1} \stackrel{\text{def}}{=} \mathcal{S}_m[\precsim^{\flat}]. \end{array}$$

By simple induction proofs one can show that $\mathcal{R}_\omega$ and $\mathcal{S}_\omega$ are preserved under renaming and substitution and satisfy

$$\mathcal{S}_\omega \subseteq \mathcal{R}_\omega^{\flat}. \tag{3}$$

The main step in the proof is to establish

$$\mathcal{R}_\omega \subseteq \mathcal{B}(\mathcal{S}_\omega). \tag{4}$$

Expanding the definitions, what we need to prove is:

$$\begin{array}{l} \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash c \ \mathcal{R}_m \ c', \\ \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash c[v] \rightsquigarrow^n \xrightarrow{a} u \mid w \end{array}$$

imply there exists u', w' such that

$$\begin{array}{l} \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash c'[v'] \rightsquigarrow^* \xrightarrow{a} u' \mid w', \\ \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}}, \Delta \vdash u, w \ \ \mathcal{S}_\omega \ \ u', w'. \end{array}$$

The proof is by induction on n and m, ordered lexicographically. The argument is analogous to the proof of the untyped substitution lemma in [13]. The type identifiers and opaque values add some extra bookkeeping, using Lemma 2 and some renaming properties of ultimate patterns, but the underlying argument is concerned with the invocation of functions (the labelled transitions).

The inclusions (4), (3), and (2) imply that $\mathcal{R}_\omega$ and $\mathcal{S}_\omega$ are active and passive bisimulations, as required. □

We now define bisimilarity on commands and values, written $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \precsim M'$ and $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V \precsim V'{:}B$. First in flat type contexts:

- $\Delta \vdash^{\mathsf{n}} M \precsim M'$ if $\Delta \parallel \emptyset \vdash \epsilon; M \precsim \epsilon; M'$.
- $\Delta \vdash^{\mathsf{v}} V \precsim V' : B$ if $V = p[w]$ and $V' = p[w']$ for some ultimate value pattern $\Delta \parallel \emptyset \mid \Delta_{\mathsf{o}} \vdash p : B$ and flat substitutions w, w' such that $\Delta \parallel \Delta_{\mathsf{o}} \vdash w \precsim^{\flat} w'$.

This is an equivalence relation on terms (commands and values), in flat type contexts, because bisimilarity is an equivalence on (active and passive) LTS states. Furthermore, it is easy to see that it includes the β-laws (the C-machine transitions in Fig. 2), in flat type contexts. The η-law for functions,

$$\overrightarrow{\mathtt{X}}, \mathtt{f}{:}\neg A \vdash^{\mathsf{v}} \mathtt{f} \precsim \lambda\mathtt{x}.\mathtt{f}\mathtt{x} : \neg A, \ \text{ if } \overrightarrow{\mathtt{X}} \vdash A,$$

holds because

$$\begin{array}{l} \{(\overrightarrow{\mathtt{X}}, \mathtt{f}{:}\neg A, \Delta_{\mathsf{i}} \parallel \mathtt{g}{:}\neg A, \Delta_{\mathsf{o}} \vdash (\mathtt{f}/\mathtt{g}, u), (\lambda\mathtt{x}.\mathtt{f}\mathtt{x}/\mathtt{g}, u)) \mid \\ \ \overrightarrow{\mathtt{X}}, \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u\} \end{array}$$

is a passive bisimulation.

Example 9 *In Example 7 we decomposed two semaphores into the same ultimate value pattern and into flat substitutions which were then related by a passive bisimulation in Example 8. Thus*

$$\emptyset \vdash^{\mathsf{v}} \mathtt{semaphore1} \precsim \mathtt{semaphore2} : \mathtt{Semaphore}.$$

We extend bisimilarity to general type contexts by "flattening" bisimilarity judgements in general type contexts $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}$ to conjunctions of bisimilarity judgements in flat contexts Δ:

- $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \precsim M'$ if

$$\begin{array}{l} \forall \overrightarrow{\Delta, p}.\ \emptyset \parallel \overrightarrow{\mathtt{X}} \mid \overrightarrow{\Delta} \vdash \overrightarrow{p : A} \\ \qquad \Rightarrow \overrightarrow{\mathtt{X}}, \overrightarrow{\Delta} \vdash^{\mathsf{n}} M[\overrightarrow{p/\mathtt{x}}] \precsim M'[\overrightarrow{p/\mathtt{x}}]. \end{array} \tag{5}$$

- $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V \precsim V' : B$ if

$$\begin{array}{l} \forall \overrightarrow{\Delta, p}.\ \emptyset \parallel \overrightarrow{\mathtt{X}} \mid \overrightarrow{\Delta} \vdash \overrightarrow{p : A} \\ \qquad \Rightarrow \overrightarrow{\mathtt{X}}, \overrightarrow{\Delta} \vdash^{\mathsf{v}} V[\overrightarrow{p/\mathtt{x}}] \precsim V'[\overrightarrow{p/\mathtt{x}}] : B. \end{array} \tag{6}$$

It is easy to see that reflexivity, transitivity, symmetry, and the β-laws and η-law generalize to bisimilarity in general type contexts; for the β-laws this follows from the fact that β-reductions are preserved by term and type substitutions. Now we can also show the η-laws for other types, listed in Fig. 5. When we expand definition (5), the η-laws become β-laws. For instance, consider the last η-law, for existential types. We need to show

$$\begin{array}{l} \overrightarrow{\mathtt{X}}, \mathtt{x}_1{:}A_1 \ldots, \Delta, \ldots \mathtt{x}_n{:}A_n \vdash^{\mathsf{n}} \\ \quad (\mathtt{pm}\ \mathtt{x}_{\hat{\imath}}\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y} \rangle . M[\langle \mathtt{Y}, \mathtt{y} \rangle / \mathtt{x}_{\hat{\imath}}])[p/\mathtt{x}_{\hat{\imath}}] \precsim M[p/\mathtt{x}_{\hat{\imath}}] \end{array}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \quad A_{\hat{\imath}} = 1}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M[\langle\rangle/\mathtt{x}_{\hat{\imath}}] \approx M}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \quad A_{\hat{\imath}} = B_1 \times B_2}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ \mathtt{x}_{\hat{\imath}}\ \mathtt{as}\ \langle \mathtt{y}_1, \mathtt{y}_2\rangle.M[\langle \mathtt{y}_1, \mathtt{y}_2\rangle/\mathtt{x}_{\hat{\imath}}] \approx M}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}, \vdash^{\mathsf{n}} M \quad A_{\hat{\imath}} = \sum_{i\in I} B_i}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ \mathtt{x}_{\hat{\imath}}\ \mathtt{as}\ \{\langle i, \mathtt{y}_i\rangle.M[\langle i, \mathtt{y}_i\rangle/\mathtt{x}_{\hat{\imath}}]\}_{i\in I} \approx M}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \quad A_{\hat{\imath}} = \neg B}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M[\lambda\mathtt{y}.\mathtt{x}_{\hat{\imath}}\mathtt{y}/\mathtt{x}_{\hat{\imath}}] \approx M}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \quad A_{\hat{\imath}} = \mu\mathtt{Y}.B}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ \mathtt{x}_{\hat{\imath}}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{y}.M[\mathtt{fold}\ \mathtt{y}/\mathtt{x}_{\hat{\imath}}] \approx M}$$

$$\frac{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \quad A_{\hat{\imath}} = \exists\mathtt{Y}.B}{\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ \mathtt{x}_{\hat{\imath}}\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M[\langle \mathtt{Y}, \mathtt{y}\rangle/\mathtt{x}_{\hat{\imath}}] \approx M}$$

Figure 5. JWA η-laws

for all Δ, p such that $\emptyset \parallel \overrightarrow{\mathtt{X}} \mid \Delta \vdash p : \exists\mathtt{Y}.B$. According to the ultimate value pattern definition in Fig. 4, $\Delta = \mathtt{Y}, \Delta'$ and $p = \langle \mathtt{Y}, p'\rangle$ for some $\emptyset \parallel \overrightarrow{\mathtt{X}}, \mathtt{Y} \mid \Delta' \vdash p' : B$. When we substitute $\langle \mathtt{Y}, p'\rangle$ for p in the equation, it turns into the β-law for existential values.

Also note that if $\overrightarrow{\mathtt{X}} \vdash A_{\hat{\imath}}$ is an empty type, meaning that the set of patterns $\emptyset \parallel \overrightarrow{\mathtt{X}} \mid \Delta \vdash p : A_{\hat{\imath}}$ is empty, then

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M \approx M', \text{ if } \overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} M, M', \qquad (7)$$

hold vacuously by the definition (5).

Example 10 *List reversal is an involution,*

$$\mathtt{X}, \mathtt{a}{:}\mathtt{List}(\mathtt{X}), \mathtt{k}{:}\neg\mathtt{List}(\mathtt{X}) \vdash^{\mathsf{n}} \mathtt{reverse}\langle \mathtt{X}, \langle \mathtt{a}, \lambda\mathtt{b}.\mathtt{reverse}\langle \mathtt{X}, \langle \mathtt{b}, \mathtt{k}\rangle\rangle\rangle\rangle \approx \mathtt{k}\mathtt{a}.$$

To prove this, we must show

$$\mathtt{X}, \Delta, \mathtt{k}{:}\neg\mathtt{List}(\mathtt{X}) \vdash^{\mathsf{n}} \mathtt{reverse}\langle \mathtt{X}, \langle p, \lambda\mathtt{b}.\mathtt{reverse}\langle \mathtt{X}, \langle \mathtt{b}, \mathtt{k}\rangle\rangle\rangle\rangle \approx \mathtt{k}\,p, \qquad (8)$$

for every ultimate value pattern $\emptyset \parallel \mathtt{X} \mid \Delta \vdash p : \mathtt{List}(\mathtt{X})$. Recall from Example 5 that each such p is a list of fresh opaque values. We can prove (8) by induction on the length of the list p.

Lemma 5 *Bisimilarity on terms is substitutive.*

Proof outline Follows from Lemma 4, using Lemma 2 and renaming properties of ultimate patterns as in the proof of Lemma 4. □

Proposition 6 *Bisimilarity on terms is a congruence.*

Proof We must prove that bisimilarity is preserved by every typing rule in Fig. 1.

In the case of $\neg$-introduction, this means that

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}, \mathtt{y}{:}B \vdash^{\mathsf{n}} M \approx M'$$

implies

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} \lambda\mathtt{y}.M \approx \lambda\mathtt{y}.M' : \neg B.$$

The argument requires a lemma about preservation of $\approx^\flat$ under concatenation of passive states. It is analogous to the proof in [23].

All the other cases follow easily from the fact that bisimilarity is an equivalence relation, the β-laws, and substitutivity, Lemma 5. For instance, in the case of $\exists$-elimination, consider arbitrary ultimate value patterns

$$\emptyset \parallel \overrightarrow{\mathtt{X}} \mid \Delta_j \vdash p_j : A_j, \text{ for } 1 \le j \le |\overrightarrow{\mathtt{x}}|,$$

and observe that any ultimate value pattern

$$\emptyset \parallel \overrightarrow{\mathtt{X}} \mid \Theta \vdash q : \exists\mathtt{Y}.B$$

has the form $q = \langle \mathtt{Y}, q'\rangle$ with $\Theta = \mathtt{Y}, \Theta'$ and

$$\emptyset \parallel \overrightarrow{\mathtt{X}}, \mathtt{Y} \mid \Theta' \vdash q' : B.$$

Moreover,

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\Delta}, \mathtt{Y}, \Theta' \vdash^{\mathsf{n}} M[\overrightarrow{p/\mathtt{x}}, q'/\mathtt{y}] \approx M'[\overrightarrow{p/\mathtt{x}}, q'/\mathtt{y}]$$

implies

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\Delta}, \mathtt{Y}, \Theta' \vdash^{\mathsf{n}} (\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M)[\overrightarrow{p/\mathtt{x}}, q/\mathtt{z}] \approx (\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M')[\overrightarrow{p/\mathtt{x}}, q/\mathtt{z}]$$

by β-equality and transitivity. Hence

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A}, \mathtt{z}{:}\exists\mathtt{Y}.B \vdash^{\mathsf{n}} \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M \approx \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M'$$

and the result

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M \approx \mathtt{pm}\ V'\ \mathtt{as}\ \langle \mathtt{Y}, \mathtt{y}\rangle.M'$$

follows from the premiss $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}A} \vdash^{\mathsf{v}} V \approx V' : \exists\mathtt{Y}.B$ and substitutivity, Lemma 5 □

Bisimilarity is non-trivial in the sense that it does not relate all well-typed terms. More specifically, it is immediate from the definition of bisimilarity that it is *computationally adequate* in the sense that no diverging command (one with

an infinite reduction sequence) is bisimilar to a terminating command (one that reduces to a terminal command).

We can define *contextual equivalence* to be the greatest computationally adequate congruence relation (cf. [11, Sect. 3.7]). Since bisimilarity is both computational adequate and a congruence, it is included in contextual equivalence.

6 Type Isomorphisms

To illustrate the bisimulation proof principle we now prove two type isomorphisms, up to bisimilarity, for existential types.

A type isomorphism between two types $\overrightarrow{\mathtt{X}} \vdash A, A'$ consists of two functions $F_1{:}\neg(A\times\neg A'), F_2{:}\neg(A'\times\neg A)$ that are mutually inverse:

$$\mathtt{x}{:}A, \mathtt{k}{:}\neg A' \vdash^{\mathsf{n}} F_1\langle \mathtt{x}, \lambda \mathtt{z}.F_2\langle \mathtt{z}, \mathtt{k}\rangle\rangle \approx \mathtt{kx},$$
$$\mathtt{x}{:}A', \mathtt{k}{:}\neg A \vdash^{\mathsf{n}} F_2\langle \mathtt{x}, \lambda \mathtt{z}.F_1\langle \mathtt{z}, \mathtt{k}\rangle\rangle \approx \mathtt{kx}.$$

We place the additional requirement that F_1 and F_2 are "effect-free" in the sense that

$$\mathtt{x}{:}A, \mathtt{j}{:}\neg\neg\neg A' \vdash^{\mathsf{n}} \mathtt{j}\ \lambda\mathtt{k}.F_1\langle \mathtt{x}, \mathtt{k}\rangle \approx F_1\langle \mathtt{x}, \lambda \mathtt{z}.\mathtt{j}\lambda\mathtt{k}.\mathtt{kz}\rangle,$$
$$\mathtt{x}{:}A', \mathtt{j}{:}\neg\neg\neg A \vdash^{\mathsf{n}} \mathtt{j}\ \lambda\mathtt{k}.F_2\langle \mathtt{x}, \mathtt{k}\rangle \approx F_2\langle \mathtt{x}, \lambda \mathtt{z}.\mathtt{j}\lambda\mathtt{k}.\mathtt{kz}\rangle$$

(the CPS equivalent of Führmann's *thunkability* [6]). This requirement holds trivially for all our examples.

Example 11 *If $\mathtt{X}, \mathtt{Y} \vdash A$ then $\exists\mathtt{X}.\exists\mathtt{Y}.A$ is isormorphic to $\exists\mathtt{Y}.\exists\mathtt{X}.A$, because $F_1 = F_2 = \lambda\langle\langle \mathtt{X}, \langle \mathtt{Y}, \mathtt{x}\rangle\rangle, \mathtt{k}\rangle.\mathtt{k}\langle \mathtt{Y}, \langle \mathtt{X}, \mathtt{x}\rangle\rangle$ form an isomorphism. The proof that they are mutual inverses (and effect-free) follows from the η-law for existentials and β-conversion, or directly from the definition (5).*

Example 12 *The type $\exists\mathtt{X}.\mathtt{X}$ is isomorphic to 1, because*

$$F_1 = \lambda\langle\langle \mathtt{X}, \mathtt{x}\rangle, \mathtt{k}\rangle.\mathtt{k}\langle\rangle, \quad F_2 = \lambda\langle\langle\rangle, \mathtt{k}\rangle.\mathtt{k}\langle 1, \langle\rangle\rangle,$$

form an isomorphism. The interesting equation is:

$$\mathtt{z}{:}\exists\mathtt{X}.\mathtt{X}, \mathtt{k}{:}\neg\exists\mathtt{X}.\mathtt{X} \vdash^{\mathsf{n}} F_1\langle \mathtt{z}, \lambda\mathtt{z}.F_2\langle \mathtt{z}, \mathtt{k}\rangle\rangle \approx \mathtt{kz}.$$

By definition (5) and β-conversion, we must prove

$$\mathtt{X}, \mathtt{x}{:}\mathtt{X}, \mathtt{k}{:}\neg\exists\mathtt{X}.\mathtt{X} \parallel \emptyset \vdash \epsilon; \mathtt{k}\langle 1, \langle\rangle\rangle \approx \epsilon; \mathtt{k}\langle \mathtt{X}, \mathtt{x}\rangle.$$

Each side has the output transition:

$$\Delta_{\mathsf{i}} \parallel \emptyset \mid \mathtt{Y}, \mathtt{y}{:}\mathtt{Y} \vdash \epsilon; \mathtt{k}\langle 1, \langle\rangle\rangle \xrightarrow{\mathtt{k}\langle \mathtt{Y}, \mathtt{y}\rangle} \epsilon \mid 1/\mathtt{Y}, \langle\rangle/\mathtt{y},$$
$$\Delta_{\mathsf{i}} \parallel \emptyset \mid \mathtt{Y}, \mathtt{y}{:}\mathtt{Y} \vdash \epsilon; \mathtt{k}\langle \mathtt{X}, \mathtt{x}\rangle \xrightarrow{\mathtt{k}\langle \mathtt{Y}, \mathtt{y}\rangle} \epsilon \mid \mathtt{X}/\mathtt{Y}, \mathtt{x}/\mathtt{y}.$$

where $\Delta_{\mathsf{i}} = \mathtt{X}, \mathtt{x}{:}\mathtt{X}, \mathtt{k}{:}\neg\exists\mathtt{X}.\mathtt{X}$. The resulting passive states

$$\Delta_{\mathsf{i}} \parallel \mathtt{Y}, \mathtt{y}{:}\mathtt{Y} \vdash 1/\mathtt{Y}, \langle\rangle/\mathtt{y}, \quad \Delta_{\mathsf{i}} \parallel \mathtt{Y}, \mathtt{y}{:}\mathtt{Y} \vdash \mathtt{X}/\mathtt{Y}, \mathtt{x}/\mathtt{y},$$

have no output functions, so they cannot receive any inputs (there are no input labels) and are thus vacuously bisimilar.

One can extrapolate the previous example to show that any two closed types with equally many ultimate value patterns (their sets of ultimate value patterns have the same cardinality) are isomorphic provided none of their patterns has any function holes.

Example 13 *The type $\exists\mathtt{X}.\neg\mathtt{X}$ is isomorphic to 1 with*

$$F_1 = \lambda\langle\langle \mathtt{X}, \mathtt{f}\rangle, \mathtt{k}\rangle.\mathtt{k}\langle\rangle, \quad F_2 = \lambda\langle\langle\rangle, \mathtt{k}\rangle.\mathtt{k}\langle 1, \bot\rangle,$$

where $\bot \stackrel{\text{def}}{=} \lambda\mathtt{x}.\mathtt{diverge}$. Analogously to the previous example we end up with two states with an output transition:

$$\Delta_{\mathsf{i}} \parallel \emptyset \mid \mathtt{Y}, \mathtt{g}{:}\neg\mathtt{Y} \vdash \epsilon; \mathtt{k}\langle 1, \bot\rangle \xrightarrow{\mathtt{k}\langle \mathtt{Y}, \mathtt{g}\rangle} \epsilon \mid 1/\mathtt{Y}, \bot/\mathtt{g},$$
$$\Delta_{\mathsf{i}} \parallel \emptyset \mid \mathtt{Y}, \mathtt{g}{:}\neg\mathtt{Y} \vdash \epsilon; \mathtt{k}\langle \mathtt{X}, \mathtt{f}\rangle \xrightarrow{\mathtt{k}\langle \mathtt{Y}, \mathtt{g}\rangle} \epsilon \mid \mathtt{X}/\mathtt{Y}, \mathtt{f}/\mathtt{g}.$$

where $\Delta_{\mathsf{i}} = \mathtt{X}, \mathtt{f}{:}\neg\mathtt{X}, \mathtt{k}{:}\neg\exists\mathtt{X}.\neg\mathtt{X}$. The resulting passive states

$$\Delta_{\mathsf{i}} \parallel \mathtt{Y}, \mathtt{g}{:}\neg\mathtt{Y} \vdash 1/\mathtt{Y}, \bot/\mathtt{g}, \quad \Delta_{\mathsf{i}} \parallel \mathtt{Y}, \mathtt{g}{:}\neg\mathtt{Y} \vdash \mathtt{X}/\mathtt{Y}, \mathtt{f}/\mathtt{g},$$

again have no input labels, this time because there are no ultimate value patterns of type $\mathtt{Y}$ in the requisite input-output context,

$$\mathtt{Y}, \mathtt{g}{:}\neg\mathtt{Y} \parallel \Delta_{\mathsf{i}} \mid \Delta \vdash p : \mathtt{Y},$$

as there are no output opaque values in context $\mathtt{Y}, \mathtt{g}{:}\neg\mathtt{Y}$, so the passive states are vacuously bisimilar.

With some extra effort one can extend the proof of the last example to show that $\exists\mathtt{X}.\mathtt{X}^n\times\neg\mathtt{X}$, where $\mathtt{X}^n$ is the n-fold product of $\mathtt{X}$, is isomorphic to $\sum_1^n 1$.

The last example shows that our system is significantly different from those studied in [5].

7 Genericity

As another illustration of typed normal form bisimulation reasoning about polymorphism, we show that our semantics satisfies a form of Longo, Milsted, and Soloviev's [17] equational genericity principle. It captures the idea that a generic program is unable to probe the structure of its instances.

At the level of the operational semantics, we have the following genericity property of substitution of types and opaque values, a property that is not enjoyed by substitution of functions.

Proposition 7 *Suppose $\mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W}, \Gamma \vdash^{\mathsf{n}} M$ and $\Gamma^\dagger \vdash A$ and $\Gamma[A/\mathtt{W}] \vdash \overrightarrow{V} : A$. If $\Gamma[A/\mathtt{W}] \vdash^{\mathsf{n}} M[A/\mathtt{W}, \overrightarrow{V/\mathtt{w}}] \rightsquigarrow^* T$ where T is terminal, there exists terminal $\mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W}, \Gamma \vdash^{\mathsf{n}} T'$ such that $M \rightsquigarrow^* T'$ and $T = T'[A/\mathtt{W}, \overrightarrow{V/\mathtt{w}}]$.*

Proof Induction on $\rightsquigarrow^*$ and case analysis on M. □

The equational genericity principle is a corresponding property for program equivalence: generic commands are equivalent if they are equivalent at any given instance. This principle is not valid for arbitrary instantiations. For instance,

$$\mathtt{W}, \mathtt{w}_1{:}\mathtt{W}, \mathtt{w}_2{:}\mathtt{W}, \mathtt{k}{:}\neg\mathtt{W} \vdash^{\mathsf{n}} \mathtt{kw}_1 \not\approx \mathtt{kw}_2,$$

even though the equivalence holds if we substitute 1 for $\mathtt{W}$,

$$\mathtt{w}_1{:}1, \mathtt{w}_2{:}1, \mathtt{k}{:}\neg 1 \vdash^{\mathsf{n}} \mathtt{kw}_1 \approx \mathtt{kw}_2.$$

A closed type A is *generic* for $\approx$ when, for every $\mathtt{W}$, $\Gamma \vdash^{\mathsf{n}} M, M'$, if $\Gamma[A/\mathtt{W}] \vdash^{\mathsf{n}} M[A/\mathtt{W}] \approx M'[A/\mathtt{W}]$ then $M \approx M'$. Generic types have been considered in [2, 17].

For illustration, we now show that `Bool` is a generic type. (The proof can be adapted to many other closed types, but a more exhaustive investigation of generic types is beyond the scope of this paper).

Proposition 8 *The type* `Bool` *is generic for* $\approx$.

Proof For any active or passive relation $\mathcal{X}$, we write $\mathcal{G}(\mathcal{X})$ for the active or passive relation

$$\{(\mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W}, \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash x, x') \mid \Delta_{\mathsf{i}}[\mathtt{Bool}/\mathtt{W}] \parallel \Delta_{\mathsf{o}}[\mathtt{Bool}/\mathtt{W}] \vdash x[k]\ \mathcal{X}\ x'[k], \text{ for each boolean substitution } \emptyset \parallel \mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W} \vdash k = \mathtt{Bool}/\mathtt{W}, \overrightarrow{U/\mathtt{w}}\}.$$

The proposition follows straightforwardly if $\mathcal{G}(\approx)$ is a bisimulation, hence is contained in $\approx$.

Suppose

$$\mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W}, \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \vdash u; M\ \mathcal{G}(\approx)\ u'; M' \tag{9}$$

and $M \rightsquigarrow^* \mathtt{f}p[w]$ where p is an ultimate value pattern and w is the filling. Then, writing k_0 for the boolean substitution $\overrightarrow{\mathtt{Bool}/\mathtt{W}, \mathtt{true}/\mathtt{w}}$, we have $M[k_0] \rightsquigarrow^* \mathtt{f}p[k_0][w[k_0]]$. It is easy to see that $p[k_0]$ is an ultimate value pattern and (9) gives us $M'[k_0] \rightsquigarrow^* \mathtt{f}p[k_0][w'']$ for some w''. Prop. 7 then gives us $M' \rightsquigarrow^* \mathtt{f}V$ for some V such that $V[k_0] = p[k_0][w'']$. Since the LTS is deterministic, V is unique; we decompose it as $p'[w']$.

Next we prove a lemma: for any ultimate value patterns $\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x}{:}\Xi}, \mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W} \parallel \overrightarrow{\mathtt{Y}} \mid \Delta \vdash p, p' : D$ such that $p[k] = p'[k]$ for each boolean substitution $k = \mathtt{Bool}/\mathtt{W}, \overrightarrow{U/\mathtt{w}}$, we have $p = p'$. This is proved by induction on p.

Back to our proof. For any boolean substitution k we have $M[k] \rightsquigarrow^* \mathtt{f}p[k][w[k]]$ and $M'[k] \rightsquigarrow^* \mathtt{f}p'[k][w'[k]]$, where $\mathtt{W}, \overrightarrow{\mathtt{w}}{:}\mathtt{W}, \Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}} \mid \Delta \vdash \mathtt{f}p, \mathtt{f}p'$ and $p[k], p'[k]$ are ultimate value patterns. Then (9) gives us $p[k] = p'[k]$ and

$$\Delta_{\mathsf{i}}[\mathtt{Bool}/\mathtt{W}] \parallel \Delta_{\mathsf{o}}[\mathtt{Bool}/\mathtt{W}], \Delta[\mathtt{Bool}/\mathtt{W}] \vdash u[k], w[k] \approx^{\flat} u'[k], w'[k].$$

The lemma gives us $p = p'$ and by definition

$$\Delta_{\mathsf{i}} \parallel \Delta_{\mathsf{o}}, \Delta \vdash u, w\ \mathcal{G}(\approx^{\flat})\ u', w'.$$

Thus $\mathcal{G}(\approx) \subseteq \mathcal{B}(\mathcal{G}(\approx^{\flat}))$.

It is straightforward to show that $\mathcal{G}(\approx^{\flat}) \subseteq \mathcal{G}(\approx)^{\flat}$, hence $\mathcal{G}(\approx) \subseteq \mathcal{B}(\mathcal{G}(\approx)^{\flat})$, which means that $\mathcal{G}(\approx)$ is a bisimulation. □

Acknowledgements We thank Radha Jagadeesan for directing our attention to the genericity theorem.

References

[1] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *13th LICS*, pages 334–344. IEEE, 1998.

[2] S. Abramsky and R. Jagadeesan. A game semantics for generic polymorphism. *Ann. Pure Appl. Logic*, 133(1-3):3–37, 2005.

[3] A. J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *15th ESOP*, volume 3924 of *LNCS*, pages 69–83, 2006.

[4] M. Berger, K. Honda, and N. Yoshida. Genericity and the Pi-Calculus. *Acta Informatica*, 42(2):83–141, December 2005.

[5] J. de Lataillade. *Quantification du second ordre en smantique des jeux - Application aux isomorphismes de types*. PhD thesis, Université Paris Diderot (Paris 7), 2007.

[6] C. Führmann. Direct models of the computational lambda-calculus. *ENTCS*, 20:147–172, 1999.

[7] A. D. Gordon. Operational equivalences for untyped and polymorphic object calculi. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*. Cambridge U.P., 1998.

[8] D. J. D. Hughes. Games and definability for system F. In *12th LICS*, pages 76–86. IEEE, 1997.

[9] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Inf. and Comp.*, 163(2):285–408, 2000.

[10] J. Laird. A fully abstract trace semantics for general references. In *34th ICALP*, volume 4596 of *LNCS*, pages 667–679. Springer, 2007.

[11] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, Dec. 1998. BRICS Dissertation Series DS-98-2.

[12] S. B. Lassen. Eager normal form bisimulation. In *20th LICS*, pages 345–354. IEEE, 2005.

[13] S. B. Lassen. Head normal form bisimulation for pairs and the $\lambda\mu$-calculus (extended abstract). In *21st LICS*, pages 297–306. IEEE, 2006.

[14] S. B. Lassen and P. B. Levy. Typed normal form bisimulation. In *16th CSL*, volume 4646 of *LNCS*, pages 283–297. Springer, 2007.

[15] P. B. Levy. Game semantics using function inventories. Talk given at *Geometry of Computation 2006*, Marseille, 2006.

[16] P. B. Levy. *Call-By-Push-Value. A Functional/Imperative Synthesis*. Semantic Struct. in Computation. Springer, 2004.

[17] G. Longo, K. Milsted, and S. Soloviev. The genericity theorem and parametricity in the polymorphic λ-calculus. *Theoretical Computer Science*, 121(1–2):323–349, 6 Dec. 1993.

[18] A. Murawski and L. Ong. Evolving games and essential nets for affine polymorphism. In *TLCA*, volume 2044 of *LNCS*. Springer, 2001.

[19] B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *J. ACM*, 47(3):531–584, 2000.

[20] A. M. Pitts. Typed operational reasoning. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7, pages 245–289. The MIT Press, 2005.

[21] C. Queinnec. *Lisp in Small Pieces*. Cambridge U.P., 1996.

[22] J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing*, 83:513–523, 1983.

[23] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *34th POPL*, pages 63–74. ACM, 2007.

[24] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *J. ACM*, 54(5), 2007.

[25] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.