Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 276 (2011) 215–235

www.elsevier.com/locate/entcs

From Applicative to Environmental Bisimulation[1]

Vasileios Koutavas[2,5]

Trinity College Dublin, Ireland

Paul Blain Levy[3,6]

University of Birmingham, U.K.

Eijiro Sumii[4]

Tohoku University, Japan

Abstract

We illuminate important aspects of the semantics of higher-order functions that are common in the presence of local state, exceptions, names and type abstraction via a series of examples that add to those given by Stark. Most importantly we show that any of these language features gives rise to the phenomenon that certain behaviour of higher-order functions can only be observed by providing them with arguments which internally call the functions again. Other examples show the need for the observer to accumulate values received from the program and generate new names. This provides evidence for the necessity of complex conditions for functions in the definition of environmental bisimulation, which deviates in each of these ways from that of applicative bisimulation.

Keywords: Environmental bisimulation, applicative bisimulation, local state, existential types.

1 Introduction

Applicative bisimulation [1] provides a relational semantics that elegantly encodes extensionality of higher-order functions. It has been shown to be sound and complete with respect to contextual equivalence for pure lambda calculi [1,8], object

[1] Preliminary report appeared in the Dagstuhl Seminar Proceedings 10351 [13].

[2] Email: Vasileios.Koutavas@scss.tcd.ie

[3] Email: P.B.Levy@cs.bham.ac.uk

[4] Email: sumii@ecei.tohoku.ac.jp

[5] Supported by SFI project SFI 06 IN.1 1898.

[6] Supported by EPSRC Advanced Research Fellowship EP/E056091/1.

doi:10.1016/j.entcs.2011.09.023

calculi [7], and languages with I/O [33]. However, several recent bisimulation semantics of languages (stateful and pure) with higher-order functions are based on *environmental bisimulation* [31,32,27,15,5,14,16], a definition with significantly more complex conditions. Despite its applicability to numerous higher-order languages, the additional complexity of environmental bisimulation has not been sufficiently justified. The question we ask in this paper (which we answer in the affirmative) is whether this complexity is necessary.

Both applicative and environmental bisimulation test related functions by applying them to arguments, and require that the two applications equi-terminate and give related results. However, there are two ways in which environmental bisimulation deviates from applicative bisimulation, each by giving a degree of freedom to the context in the tests that it can make.

- *Accumulation of values:* In applicative bisimulation, the context may call only related functions it has just received from the related programs. In environmental bisimulation, it accumulates the functions it has received, so it can call them at any time, possibly multiple times.
- *Resourceful arguments:* In applicative bisimulation, the context supplies a single *closed* value to which related functions get applied. But in environmental bisimulation it supplies a single *open* value, closed by corresponding functions from the inventory, and therefore the related functions get applied to different arguments.

In a stateful language (or a language with names), the necessity of the first deviation is quite plausible. Accumulation is needed because a function may return a different value the second time it is applied. But the necessity of resourceful arguments appears more questionable. Even more so in a language with exceptions and polymorphism.

A related point was made by Mason and Talcott [22] in their study of a relational theory for a language with general store. They gave an example inequivalence showing the need for the opponent to assign to global locations before each function application [22, Sec. 3]. Therefore they identified the need for extending the definition of applicative bisimulation to apply to effectful languages:

> "[Applicative] bisimulation provides an alternative approach to equivalence and deserves consideration in computation systems that permit effects other than nontermination. The definition of bisimulation relation assumes that extensionality is consistent. Since the presence [of] memory effects makes this no longer true, the basic definition would require some modification in order to extend the methods of Abramsky and Howe to the computational language presented in this paper. We plan to investigate this approach."

In this paper we investigate two versions of environmental bisimulations, one with no accumulation and one without resourceful arguments. We emphasize the bisimulation without resourceful arguments since it is the more subtle. We show by examples that these bisimulations are unsound in an array of languages that use state, exceptions, names, and polymorphism. In this way we identify common aspects in the behaviour of higher-order functions in all of these languages, and

justify the complexity in the definition of environmental bisimulation.

Our examples for exceptions and polymorphism are, we believe, the first in the literature that invalidate the defective bisimulations in these languages. However, our examples for state and names are not the first of their kind; Stark gave examples that embody the same principles [28, pp. 24–25, examples 11 and 14, and p. 104, examples 12 and 14].

Additionally, in this paper we investigate the provision in the definition of environmental bisimulation for the context to create new names at every step of the bisimulation, instead of only generating an arbitrary amount of names at the beginning. For deterministic languages we believe this is unnecessary. In the presence of nondeterminism, however, there is no single answer. We give novel examples that show the necessity of this provision for environmental bisimulation to be sound with respect to must-testing, in the presence of countable nondeterminism, and with respect to lower bisimilarity, in the presence of even finite nondeterminism. We believe that name generation at every step is unnecessary for may testing and, in the presence of finite nondeterminism, for must testing.

We start by examining a pure language (Sec. 2), for which we define applicative and environmental bisimulation. We then study a language with general state (Sec. 3). In this setting, we define the two defective versions of environmental bisimulation, and present simple examples that show their unsoundness with respect to contextual equivalence. We then present similar examples for a language with exceptions (Sec. 4.1), names (Sec. 4.2), and existential types (Sec. 4.3). Finally, we present examples that show the unsoundness of fixed name-set bisimulation in the cases mentioned above (Sec. 5).

2 Bisimulations in a Pure Language

2.1 The Language λ

To understand the two deviations from applicative bisimulation, discussed in the introduction, we first review applicative and environmental bisimulation in a pure setting. We choose a call-by-value λ-calculus with recursion, which we call λ and will serve as the basis for the languages we study later on in this paper. The types of λ are given by

$$A, B, C ::= \quad 0 \mid A + A \mid 1 \mid A \times A \mid A \to A \mid \mathtt{X} \mid \mathtt{rec\ X}.\ A$$

We use the syntax of "fine-grain call-by-value", in which values and computations are distinguished, and returning and sequencing (`to`) are made explicit:

$$\begin{aligned} V ::= \ & \mathtt{x} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \langle\rangle \mid \langle V, V\rangle \mid \lambda \mathtt{x}.\, M \mid \mathtt{rec\ f}\ \lambda \mathtt{x}.\, M \mid \mathtt{fold}\ V \\ M ::= \ & \mathtt{return}\ V \mid M\ \mathtt{to\ x}.\ M \mid V\ V \mid \mathtt{match}\ V\ \mathtt{as}\ \{\} \\ & \mid \mathtt{match}\ V\ \mathtt{as}\ \{\mathtt{inl\ x}.\ M, \mathtt{inr\ x}.\ M\} \mid \mathtt{match}\ V\ \mathtt{as}\ \langle\rangle.\ M \\ & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M \mid \mathtt{match}\ V\ \mathtt{as\ fold\ x}.\ M \end{aligned}$$

We have typing judgements $\Gamma \vdash M : A$ and for values $\Gamma \vdash^{\mathsf{v}} V : A$, where Γ is a list of distinct closed-typed identifiers and A is a closed type, defined in the usual inductive way. We write $\Gamma \vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{A}$ to mean $\Gamma \vdash^{\mathsf{v}} V_i : A_i$, for all $i < |\overrightarrow{A}|$. We abbreviate $\mathtt{bool} \stackrel{\text{def}}{=} 1 + 1$ with $\mathtt{true}$ and $\mathtt{false}$ defined accordingly, abbreviate $\mathtt{diverge} \stackrel{\text{def}}{=} (\mathtt{rec\ f}\ \lambda \mathtt{x.\,fx})\langle\rangle$ and write $M\ \mathtt{to\ x}.\ N$ as $M; N$ when $\mathtt{x}$ does not occur in N. We then define as usual $M \Downarrow_B V$, for $\vdash M : B$ and $\vdash^{\mathsf{v}} V : B$.

2.2 Ultimate Pattern Matching

It is useful to note that any closed value consists of tags and functions. The tags constitute an *ultimate pattern* [18], and the functions constitute the *filling* of the pattern. For example, we divide the value

$$\langle \mathtt{inl}\ \lambda \mathtt{x}.M, \mathtt{inr}\ \langle \mathtt{inl}\ \lambda \mathtt{y}.N, \langle\rangle\rangle\rangle$$

into the ultimate pattern $\langle \mathtt{inl}\ -, \mathtt{inr}\ \langle \mathtt{inl}\ -, \langle\rangle\rangle\rangle$ and the filling $\lambda \mathtt{x}.M, \lambda \mathtt{y}.N$.

To make this precise, we define for each type A a set $\mathsf{ulpatt}(A)$ of ultimate patterns p, each equipped with a list $H(p)$ of function types. These sets are defined by induction:

- $-_{A \to B} \in \mathsf{ulpatt}(A \to B)$ and $H(-_{A \to B}) \stackrel{\text{def}}{=} A \to B$
- $\langle\rangle \in \mathsf{ulpatt}(1)$ and $H(\langle\rangle) \stackrel{\text{def}}{=} \varepsilon$
- if $p \in \mathsf{ulpatt}(A)$ and $p' \in \mathsf{ulpatt}(A')$ then $\langle p, p' \rangle \in \mathsf{ulpatt}(A \times A')$ and $H(\langle p, p' \rangle) \stackrel{\text{def}}{=} H(p) \cdot H(p')$
- if $p \in \mathsf{ulpatt}(A)$ then $\mathtt{inl}\ p \in \mathsf{ulpatt}(A + A')$ and $H(\mathtt{inl}\ p) \stackrel{\text{def}}{=} H(p)$
- if $p \in \mathsf{ulpatt}(A')$ then $\mathtt{inr}\ p \in \mathsf{ulpatt}(A + A')$ and $H(\mathtt{inr}\ p) \stackrel{\text{def}}{=} H(p)$
- if $p \in \mathsf{ulpatt}(A[\mathtt{rec\ X}.\ A/\mathtt{X}])$ then $\mathtt{fold}\ p \in \mathsf{ulpatt}(\mathtt{rec\ X}.\ A)$ and $H(\mathtt{fold}\ p) \stackrel{\text{def}}{=} H(p)$.

Given $p \in \mathsf{ulpatt}(A)$, and list of values $\Gamma \vdash^{\mathsf{v}} \overrightarrow{V} : H(p)$, we define a value $\Gamma \vdash^{\mathsf{v}} p[\overrightarrow{V}] : A$ in the obvious way. Unique decomposition is immediate:

Theorem 2.1 *For any closed value $\vdash^{\mathsf{v}} V : A$, there is unique $p \in \mathsf{ulpatt}(A)$ and list of closed values $\vdash^{\mathsf{v}} \overrightarrow{W} : H(p)$ such that $V = p[\overrightarrow{W}]$.*

2.3 Applicative Bisimulation

We now define our applicative bisimulation. It will be convenient for later extensions of this definition to give it as a set of tuples which we call *relatees*. A relatee is a tuple $(\overrightarrow{A \to B}; \overrightarrow{V}; \overrightarrow{V'})$ consisting of

- a list of function types $\overrightarrow{A \to B}$
- a list of functions $\vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{A \to B}$
- a list of functions $\vdash^{\mathsf{v}} \overrightarrow{V'} : \overrightarrow{A \to B}$.

These three zones of the relatee represent the public information known to the context (for this language this is only a list of types) and the two situations, $\overrightarrow{V}$ and $\overrightarrow{V'}$, that we want to relate.

The conditions of applicative bisimulation say that when we apply corresponding functions to the same closed value, the applications equi-terminate and the resulting values have the same ultimate pattern with bisimilar fillings.

Definition 2.2 A set $\mathcal{R}$ of relatees is an *applicative bisimulation* when $(\overrightarrow{A \to B}; \overrightarrow{V}; \overrightarrow{V'}) \in \mathcal{R}$ implies that for any

- index $i < |\overrightarrow{A \to B}|$
- and closed value $\vdash^{\mathsf{v}} U : A_i$,

if $V_i\, U \Downarrow_{B_i} p[\overrightarrow{W}]$ then there exists a filling $\vdash^{\mathsf{v}} \overrightarrow{W'} : H(p)$ such that

$$V_i'\, U \Downarrow_{B_i} p[\overrightarrow{W'}] \quad \text{and} \quad (H(p); \overrightarrow{W}; \overrightarrow{W'}) \in \mathcal{R}$$

and the converse of the above condition holds when $V_i'\, U \Downarrow_{B_i} p[\overrightarrow{W'}]$.

Definition 2.3 Closed terms $\vdash M, M' : A$ are *applicatively bisimilar* when there exists applicative bisimulation $\mathbb{R}$ such that

- if $M \Downarrow_A p[\overrightarrow{W}]$ then there exists $\vdash^{\mathsf{v}} \overrightarrow{W'} : H(p)$ such that $M' \Downarrow_A p[\overrightarrow{W'}]$ and $(H(p); \overrightarrow{W}; \overrightarrow{W'})$ is contained in $\mathbb{R}$,
- the converse of the above condition holds when $M' \Downarrow_A p[\overrightarrow{W'}]$.

As we discussed in the introduction, Def. 2.2 is *non-accumulating*: the final relatee $(H(p); \overrightarrow{W}; \overrightarrow{W'})$ does not contain the functions in the starting relatee $(\overrightarrow{A \to B}; \overrightarrow{V}; \overrightarrow{V'})$. Moreover, the definition is *non-resourceful* because it applies related functions to the same closed arguments.

2.4 Environmental Bisimulation

We define environmental bisimulation for the pure language λ to illustrate its differences from applicative bisimulation. In the following section we will adapt this definition for a language with state.

Definition 2.4 A set $\mathcal{R}$ of relatees is an *environmental bisimulation* when $(\overrightarrow{A \to B}; \overrightarrow{V}; \overrightarrow{V'}) \in \mathcal{R}$ implies that for any

- index $i < |\overrightarrow{A \to B}|$
- and open value $\overrightarrow{\mathtt{f} : A \to B} \vdash^{\mathsf{v}} U : A_i$,

if $V_i\, U[\overrightarrow{V/\mathtt{f}}] \Downarrow_{B_i} p[\overrightarrow{W}]$ then there exists a filling $\vdash^{\mathsf{v}} \overrightarrow{W'} : H(p)$ such that

$$V_i'\, U[\overrightarrow{V'/\mathtt{f}}] \Downarrow_{B_i} p[\overrightarrow{W'}] \quad \text{and} \quad (\overrightarrow{A \to B} \cdot H(p); \overrightarrow{V} \cdot \overrightarrow{W}; \overrightarrow{V'} \cdot \overrightarrow{W'}) \in \mathcal{R}$$

and the converse of the above condition holds when $V_i' \, U[\overrightarrow{V'/\mathtt{f}}] \Downarrow_{B_i} p[\overrightarrow{W'}]$.

Definition 2.5 Closed terms $\vdash M, M' : A$ are *environmentally bisimilar* when there exists environmental bisimulation $\mathbb{R}$ such that

- if $M \Downarrow_A p[\overrightarrow{W}]$ then there exists $\vdash^{\mathsf{v}} \overrightarrow{W'} : H(p)$ such that $M' \Downarrow_A p[\overrightarrow{W'}]$ and $(H(p); \overrightarrow{W}; \overrightarrow{W'})$ is contained in $\mathbb{R}$,
- the converse of the above condition holds when $M' \Downarrow_A p[\overrightarrow{W'}]$.

As with applicative bisimulation, the conditions of environmental bisimulation require the applications of functions V_i and V_i', related in the originating relatee, to equi-terminate and their resulting values to have the same ultimate pattern with bisimilar fillings. However, in environmental bisimulation the argument provided to each of V_i and V_i' is constructed by closing the same open value $\overrightarrow{\mathtt{f} : A \to B} \vdash^{\mathsf{v}} U : A_i$ with the functions $\overrightarrow{V}$ and $\overrightarrow{V'}$, respectively, from the inventory of the originating relatee. This encodes the principle of *resourceful arguments*. Moreover, the concatenation of the types $\overrightarrow{A \to B} \cdot H(p)$ and the values $\overrightarrow{V} \cdot \overrightarrow{W}$ and $\overrightarrow{V'} \cdot \overrightarrow{W'}$ in the final relatee encode the *accumulation of values*.

3 Environmental Bisimulation for State

We give a stateful language and the definition of environmental bisimulation for this language. We then give two simpler versions of bisimulation, one without resourceful arguments and one without accumulation, and demonstrate their unsoundness with respect to contextual equivalence.

3.1 The language λ_s

We add to the language λ the facility to generate fresh locations that may be assigned to and read from. Rather than treating locations as values (as for names in Sec. 4.2), we use the syntax

$$M ::= \quad \cdots \mid \mathtt{l} := V.\ M \mid \mathtt{read\ l\ as\ x}.\ M \mid \mathtt{new}\ \overrightarrow{\mathtt{l} := V}.\ M$$

Thus locations can neither be stored nor returned. The typing judgements are now $\Delta; \Gamma \vdash M : A$ and $\Delta; \Gamma \vdash^{\mathsf{v}} V : A$, where Δ is a list of distinct closed-typed locations. For a state s we write $\Delta; \Gamma \vdash^{\mathsf{v}} s : \Delta'$ to mean $dom(s) = dom(\Delta')$ and $\Delta; \Gamma \vdash^{\mathsf{v}} s(i) : \Delta'(i)$. Evaluation takes the form $\Delta, s, M \Downarrow_A \Theta, t, V$, for term $\Delta; \vdash M : A$, state $\Delta; \vdash s : \Delta$, location list Θ extending Δ, value $\Theta; \vdash^{\mathsf{v}} V : A$ and state $\Theta; \vdash t : \Theta$.

The operational semantics of the term $\mathtt{new}\ \overrightarrow{\mathtt{l} := V}.\ M$ uses a gensym operation to generate fresh locations, which we call *private* since the context of the term has no direct access to them. The particular selection of the gensym operation leaves the semantics unaffected. Thus, without loss of generality, we assume that there is a countably infinite set of locations that is disjoint from the range of gensym. We call these locations *public* and use them in the following section as the domain

of the public state in our relations; i.e. the state to which the context has direct access.

3.2 Environmental Bisimulation for State

We shall write a location context as $\Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}$, where Δ_{pub} is a public location context and Δ_{priv} is a private location context, with domains the set of public and private locations, respectively. Likewise, we write a state as $s_{\mathsf{pub}} \circledast s_{\mathsf{priv}}$, and we call s_{pub} a *public state* and s_{priv} a *private state*. Note that every state is uniquely decomposable to a public and private state.

When the context calls a function, the public state provides additional communication between a function and its context, besides the argument and the result. It is essentially another argument to the function and, when the function returns, it is another result which needs to be ultimately pattern-matched. Note that this is not the case for private state since the context has no access to it.

Thus, for a public location context Δ_{pub}, we write

$$\mathsf{ulpatt}(\Delta_{\mathsf{pub}}) \stackrel{\text{def}}{=} \prod_{(\mathtt{l}:C)\in\Delta_{\mathsf{pub}}} \mathtt{l} \mapsto \mathsf{ulpatt}(C)$$

For $p \in \mathsf{ulpatt}(\Delta_{\mathsf{pub}})$, we define $H(p)$ to be the concatenation over $(\mathtt{l} : C) \in \Delta_{\mathsf{pub}}$ of $H(p(\mathtt{l}))$. Then for any $\Delta;\Gamma \vdash^{\mathsf{v}} \overrightarrow{W} : H(p)$, we define $\Delta;\Gamma \vdash^{\mathsf{v}} p[\overrightarrow{W}] : \Delta_{\mathsf{pub}}$ in the obvious way. As an example, let $\mathtt{l}_1$ and $\mathtt{l}_2$ be public locations and

$$\Delta_{\mathsf{pub}} \stackrel{\text{def}}{=} \mathtt{l}_1 : \mathtt{bool} \times \mathtt{bool},\ \mathtt{l}_2 : 1 + (\mathtt{bool} \to \mathtt{bool})$$

then the public state

$$\Delta_{\mathsf{pub}};\vdash^{\mathsf{v}} \mathtt{l}_1 \mapsto \langle \mathtt{true}, \mathtt{false}\rangle,\ \mathtt{l}_2 \mapsto \mathtt{inr}\ \lambda \mathtt{x}.\, \mathtt{read}\ \mathtt{l}_1\ \mathtt{as}\ \langle \mathtt{y}, \mathtt{z}\rangle.\ \mathtt{return}\ \mathtt{y} : \Delta_{\mathsf{pub}}$$

has ultimate pattern $p \stackrel{\text{def}}{=} \mathtt{l}_1 \mapsto \langle \mathtt{true}, \mathtt{false}\rangle,\ \mathtt{l}_1 \mapsto \mathtt{inr}\ -$.

Again, we have unique decomposition:

Theorem 3.1

(i) *For any value* $\Delta;\vdash^{\mathsf{v}} V : A$, *there is a unique* $p \in \mathsf{ulpatt}(A)$ *and* $\Delta;\vdash^{\mathsf{v}} \overrightarrow{W} : H(p)$ *such that* $V = p[\overrightarrow{W}]$.

(ii) *For any public state* $\Delta;\vdash^{\mathsf{v}} s_{\mathsf{pub}} : \Delta_{\mathsf{pub}}$, *there is a unique* $p \in \mathsf{ulpatt}(\Delta_{\mathsf{pub}})$ *and* $\Delta;\vdash^{\mathsf{v}} \overrightarrow{W} : H(p)$ *such that* $s_{\mathsf{pub}} = p[\overrightarrow{W}]$.

Environmental bisimulation for λ_s is defined over *relatees* that are tuples of the form $(\Delta_{\mathsf{pub}}, \overrightarrow{A \to B};\ \Delta_{\mathsf{priv}}, s_{\mathsf{priv}}, \overrightarrow{V};\ \Delta'_{\mathsf{priv}}, s'_{\mathsf{priv}}, \overrightarrow{V'})$, consisting of:

- a public location context Δ_{pub}
- a list of function types $\overrightarrow{A \to B}$
- a private location context Δ_{priv}

- a private state $\Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}; \vdash^{\mathsf{v}} s_{\mathsf{priv}} : \Delta_{\mathsf{priv}}$
- a list of functions $\Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{A \to B}$
- a private location context Δ'_{priv}
- a private state $\Delta_{\mathsf{pub}} \circledast \Delta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} s'_{\mathsf{priv}} : \Delta'_{\mathsf{priv}}$
- a list of functions $\Delta_{\mathsf{pub}} \circledast \Delta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{V'} : \overrightarrow{A \to B}$.

As in the applicative setting, a relatee is organized in three zones (separated by semicolons), representing public information and the two situations that we want to relate. Here the public information contains the type of the public state, besides the types of the functions. The other two zones contain the private state (and its type) that may be used in the functions.

The definition of environmental bisimulation for λ_s follows the same structure as that for λ (Def. 2.4), with the addition of the creation of a public state before the applications, which is ultimately pattern-matched at the end.

Definition 3.2 A set $\mathcal{R}$ of relatees is an *environmental bisimulation* when

$$(\Delta_{\mathsf{pub}}, \overrightarrow{A \to B}; \Delta_{\mathsf{priv}}, s_{\mathsf{priv}}, \overrightarrow{V}; \Delta'_{\mathsf{priv}}, s'_{\mathsf{priv}}, \overrightarrow{V'}) \in \mathcal{R}$$

implies that for any

- index $i < |\overrightarrow{A \to B}|$
- public location context Θ_{pub} extending Δ_{pub}
- public state $\Theta_{\mathsf{pub}}; \overrightarrow{\mathtt{f} : A \to B} \vdash^{\mathsf{v}} s_{\mathsf{pub}} : \Theta_{\mathsf{pub}}$
- and value $\Theta_{\mathsf{pub}}; \overrightarrow{\mathtt{f} : A \to B} \vdash^{\mathsf{v}} U : A_i$,

if

$$\Theta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}, s_{\mathsf{pub}}[\overrightarrow{V/\mathtt{f}}] \circledast s_{\mathsf{priv}}, V_i\, U[\overrightarrow{V/\mathtt{f}}] \Downarrow_{B_i} \Theta_{\mathsf{pub}} \circledast \Theta_{\mathsf{priv}}, q[\overrightarrow{T}] \circledast t_{\mathsf{priv}}, p[\overrightarrow{W}]$$

then there exists

- a private location context Θ'_{priv}
- a filling $\Theta_{\mathsf{pub}} \circledast \Theta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{T'} : H(q)$
- a private state $\Theta_{\mathsf{pub}} \circledast \Theta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} t'_{\mathsf{priv}} : \Theta'_{\mathsf{priv}}$
- a filling $\Theta_{\mathsf{pub}} \circledast \Theta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{W'} : H(p)$

such that

$$\Theta_{\mathsf{pub}} \circledast \Delta'_{\mathsf{priv}}, s_{\mathsf{pub}}[\overrightarrow{V'/\mathtt{f}}] \circledast s'_{\mathsf{priv}}, V'_i\, U[\overrightarrow{V'/\mathtt{f}}] \Downarrow_{B_i} \Theta_{\mathsf{pub}} \circledast \Theta'_{\mathsf{priv}}, q[\overrightarrow{T'}] \circledast t'_{\mathsf{priv}}, p[\overrightarrow{W'}]$$

and

$$(\Theta_{\mathsf{pub}}, \overrightarrow{A \to B} \cdot H(q) \cdot H(p);\ \Theta_{\mathsf{priv}}, t_{\mathsf{priv}}, \overrightarrow{V} \cdot \overrightarrow{T} \cdot \overrightarrow{W};\ \Theta'_{\mathsf{priv}}, t'_{\mathsf{priv}}, \overrightarrow{V'} \cdot \overrightarrow{T'} \cdot \overrightarrow{W'}) \in \mathcal{R} \quad (1)$$

Moreover, the converse condition holds when

$$\Theta_{\mathsf{pub}} \circledast \Delta'_{\mathsf{priv}}, s_{\mathsf{pub}}[\overrightarrow{V'/\mathtt{f}}] \circledast s'_{\mathsf{priv}}, V'_i\, U[\overrightarrow{V'/\mathtt{f}}] \Downarrow_{B_i} \Theta_{\mathsf{pub}} \circledast \Theta'_{\mathsf{priv}}, q[\overrightarrow{T'}] \circledast t'_{\mathsf{priv}}, p[\overrightarrow{W'}]$$

Definition 3.3 We say that closed terms $;\vdash M, M' : A$ are *environmentally bisimilar* when there exists an environmental bisimulation $\mathbb{R}$ such that

- if
$$\varepsilon, \varepsilon, M \Downarrow_A \Theta_{\mathsf{priv}}, t_{\mathsf{priv}}, p[\overrightarrow{W}]$$
then there exists
· a private location context Θ'_{priv}
· a private state $\Theta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} t'_{\mathsf{priv}} : \Theta'_{\mathsf{priv}}$
· a filling $\Theta'_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{W'} : H(p)$
such that
$$\varepsilon, \varepsilon, M' \Downarrow_A \Theta'_{\mathsf{priv}}, t'_{\mathsf{priv}}, p[\overrightarrow{W'}]$$
and $(\varepsilon, H(p);\ \Theta_{\mathsf{priv}}, t_{\mathsf{priv}}, \overrightarrow{W};\ \Theta'_{\mathsf{priv}}, t'_{\mathsf{priv}}, \overrightarrow{W'})$ is contained in $\mathbb{R}$,
- the converse of the above condition holds when
$$\varepsilon, \varepsilon, M' \Downarrow_A \Theta'_{\mathsf{priv}}, t'_{\mathsf{priv}}, p[\overrightarrow{W'}]$$

Def. 3.2 encompasses both deviations from applicative bisimulation discussed in the introduction; specifically:

(i) The use of *open* argument $\Theta_{\mathsf{pub}}; \overrightarrow{\mathtt{f} : A \to B} \vdash^{\mathsf{v}} U : A_i$ and state s_{pub} $\Theta_{\mathsf{pub}}; \overrightarrow{\mathtt{f} : A \to B} \vdash^{\mathsf{v}} s_{\mathsf{pub}} : \Theta_{\mathsf{pub}}$ encodes the *resourceful arguments* principle.

(ii) The concatenation of the types $\overrightarrow{A \to B} \cdot H(q) \cdot H(p)$ and the values $\overrightarrow{V} \cdot \overrightarrow{T} \cdot \overrightarrow{W}$ and $\overrightarrow{V'} \cdot \overrightarrow{T'} \cdot \overrightarrow{W'}$ in the final relatee encode the *accumulation of values.*

Dropping each of these principles leads to two alternative versions of Def. 3.2:

(i) If we require $\Theta_{\mathsf{pub}}; \vdash^{\mathsf{v}} U : A_i$ and $\Theta_{\mathsf{pub}}; \vdash^{\mathsf{v}} s_{\mathsf{pub}} : \Theta_{\mathsf{pub}}$, we say that $\mathcal{R}$ is a *closed-argument bisimulation.*

(ii) If we replace (1) by $(\Theta_{\mathsf{pub}}, H(q)\cdot H(p); \Theta_{\mathsf{priv}}, t_{\mathsf{priv}}, \overrightarrow{T}\cdot\overrightarrow{W}; \Theta'_{\mathsf{priv}}, t'_{\mathsf{priv}}, \overrightarrow{T'}\cdot\overrightarrow{W'}) \in \mathcal{R}$ we say that $\mathcal{R}$ is a *no-accumulation bisimulation.*

This also leads to two corresponding versions of relations for closed expressions. Clearly, if $;\vdash M, M' : A$ are environmentally bisimilar, then they are also closed-argument and no-accumulation bisimilar.

3.3 Resourceful Arguments and Accumulation

To show the necessity of resourceful arguments, we need to show closed-argument bisimulation unsound. The following example accomplishes that.

Example 3.4 *(Resourceful Arguments)* Consider M_1 and M'_1 of type

$(1 \to 1) \to \texttt{bool}$:

$$M_1 \stackrel{\text{def}}{=} \texttt{return}\ V_1$$
$$M_1' \stackrel{\text{def}}{=} \texttt{new flag} := \texttt{true}.\, \texttt{return}\ V_1'$$

where

$$V_1 \stackrel{\text{def}}{=} \lambda \texttt{f}{:}1 \to 1.\, \texttt{f}\ \langle\rangle;\ \texttt{return true}$$
$$V_1' \stackrel{\text{def}}{=} \lambda \texttt{f}{:}1 \to 1.\, \texttt{read flag as}$$
$$\{\texttt{true}.\ \texttt{flag} := \texttt{false};\ \texttt{f}\ \langle\rangle;\ \texttt{flag} := \texttt{true};\ \texttt{return true}$$
$$\ \texttt{false}.\ \texttt{return false}\}$$

The function V_1' returns a different value than V_1 when $\texttt{flag} = \texttt{false}$, which is only the case within the extent of the application $\texttt{f}\ \langle\rangle$. The following context distinguishes M_1 and M_1':

$$C_1 \stackrel{\text{def}}{=} \texttt{new record} := \texttt{true}.\, [\cdot]\ \texttt{to g}.$$
$$\texttt{g}\,(\lambda \texttt{x}.\, \texttt{g}\,(\lambda \texttt{y}.\, \texttt{return y})\ \texttt{to z}.\ \texttt{record} := \texttt{z};\ \texttt{return}\ \langle\rangle);$$
$$\texttt{read record as x}.\ \texttt{return x}$$

The term $C_1[M_1]$ returns `true`, while $C_1[M_1']$ returns `false`. This context is clearly resourceful: the outer argument provided to `g` contains `g` itself.

However, the following set of relatees is a closed-argument bisimulation that relates V_1 to V_1' with $\texttt{flag} \mapsto \texttt{true}$. To prove that, it suffices to show that the applications $\texttt{f}\ \langle\rangle$ in V_1 and V_1' equi-terminate in stores with related private parts. This follows from the fact that `f` and the public state are constructed by closed values and thus the applications do not depend on, and do not change, the value of the flag. Thus M_1 and M_1' are closed-argument bisimilar, and closed-argument bisimulation is unsound for λ_s.

$$\Big\{\big(\Delta_{\mathsf{pub}}, ((1 \to 1) \to \texttt{bool} \cdot \overrightarrow{A \to B});\ \Delta_{\mathsf{priv}}, s_{\mathsf{priv}}, (V_1 \cdot \overrightarrow{T});$$
$$(\texttt{flag} : \texttt{bool} \cdot \Delta'_{\mathsf{priv}}), (\texttt{flag} \mapsto \texttt{true} \cdot s_{\mathsf{priv}}\theta), (V_1' \cdot \overrightarrow{T}\theta)\big)\ \mid$$
$$\theta : \Delta_{\mathsf{priv}} \to \Delta'_{\mathsf{priv}} \text{ is a bijective renaming,}$$
$$\Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{T} : \overrightarrow{A \to B},\ \Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}; \vdash^{\mathsf{v}} s_{\mathsf{priv}} : \Delta_{\mathsf{priv}}\Big\} \qquad \square$$

We now provide an example that shows that accumulation is necessary in a bisimulation for state. Therefore no-accumulation bisimulation is unsound.

Example 3.5 *(Accumulation)* Consider M_2 and M_2' of type $1 \to \texttt{bool}$:

$$M_2 \stackrel{\text{def}}{=} \texttt{return}\ V_2$$
$$M_2' \stackrel{\text{def}}{=} \texttt{new flag} := \texttt{true}.\, \texttt{return}\ V_2'$$

where

$$V_2 \stackrel{\text{def}}{=} \lambda\langle\rangle.\,\texttt{return true}$$

$$V_2' \stackrel{\text{def}}{=} \lambda\langle\rangle.\,\texttt{read flag as } \{\texttt{true}.\ \texttt{flag} := \texttt{false};\ \texttt{return true}$$
$$\texttt{false}.\ \texttt{return false}\}$$

The function V_2' is a function that returns `true` the first time it is applied and `false` all subsequent times, whereas V_2 always returns `true`.

The following context distinguishes M_2 and M_2':

$$C_2 \stackrel{\text{def}}{=} [\cdot]\ \texttt{to f}.\ \texttt{f}\ \langle\rangle; \texttt{f}\ \langle\rangle$$

The term $C_2[M_2]$ returns `true`, while $C_2[M_2']$ returns `false`.

We can show, however, that the following set of relatees is a no-accumulation bisimulation, and therefore M_2 and M_2' are no-accumulation bisimilar. Hence no-accumulation bisimulation is unsound for λ_s.

$$\begin{array}{l}
\left\{\left(\Delta_{\mathsf{pub}}, 1 \to \texttt{bool};\ \ \varepsilon, \varepsilon, V_2;\ \ \texttt{flag} : \texttt{bool}, \texttt{flag} \mapsto \texttt{true}, V_2'\right)\right\} \cup \\
\left\{\left(\Delta_{\mathsf{pub}}, \overrightarrow{A \to B};\ \ \Delta_{\mathsf{priv}}, s_{\mathsf{priv}}, \overrightarrow{T};\right.\right. \\
\quad (\texttt{flag} : \texttt{bool} \cdot \Delta'_{\mathsf{priv}}), (\texttt{flag} \mapsto \texttt{false} \cdot s'_{\mathsf{priv}}\theta), \overrightarrow{T}\theta) \mid \\
\quad\quad \theta : \Delta_{\mathsf{priv}} \to \Delta'_{\mathsf{priv}} \text{ is a bijective renaming,} \\
\quad\quad \left.\left.\Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}; \vdash^{\mathsf{v}} \overrightarrow{T} : \overrightarrow{A \to B},\quad \Delta_{\mathsf{pub}} \circledast \Delta_{\mathsf{priv}}; \vdash^{\mathsf{v}} s_{\mathsf{priv}} : \Delta_{\mathsf{priv}}\right\}\right.
\end{array} \qquad \square$$

4 Other Language Extensions

4.1 Exceptions

We add to the λ language the facility to generate fresh exceptions that may be raised and caught. Following [4] our syntax takes the form

$$M ::= \ \cdots \mid \texttt{new e}.\ M \mid \texttt{raise e} \mid M\ \{\texttt{to x}.\ M, \overrightarrow{\texttt{catch e}.\ M_{\texttt{e}}}\}$$

To evaluate $M\ \{\texttt{to x}.\ P, \overrightarrow{\texttt{catch e}.\ M_{\texttt{e}}}\}$, we first evaluate M, and if it returns V we evaluate $P[V/\texttt{x}]$. If instead it raises an exception `e`, then if `e` appears in $\overrightarrow{\texttt{e}}$ we proceed to evaluate $M_{\texttt{e}}$, otherwise `e` remains raised. We define two big-step relations, for returning and raising respectively:

- $\Delta, M \Downarrow_A, \Theta, V$ for $\Delta; \vdash M : A$, exceptions Θ extending Δ and $\Theta; \vdash^{\mathsf{v}} V : A$.
- $\Delta, M \lightning_A \Theta, \texttt{e}$ for a term $\Delta; \vdash M : A$, exceptions Θ extending Δ and exception `e` appearing in Θ.

Ultimate pattern matching and environmental bisimulation—and its two defective variants—are defined as in Sec. 3.2.

Example 4.1 *(Resourceful arguments)* Consider M_3, M_3' of type $(1 \to 1) \to 1$:

$$M_3 \stackrel{\text{def}}{=} \texttt{new e. return } V_3$$
$$M_3' \stackrel{\text{def}}{=} \texttt{new e. return } V_3'$$

where

$$V_3 \stackrel{\text{def}}{=} \lambda \texttt{f. f}\,\langle\rangle; \texttt{ raise e}$$
$$V_3' \stackrel{\text{def}}{=} \lambda \texttt{f. f}\,\langle\rangle\,\{\texttt{to x. raise e}, \texttt{catch e. return}\,\langle\rangle\}$$

These terms are distinguished by the context

$$C_3 \stackrel{\text{def}}{=} [\cdot]\ \texttt{to g. g}\,(\lambda \texttt{x. g}\,(\lambda \texttt{y. return y}))$$

The term $C_3[M_3]$ raises an exception, while $C_3[M_3']$ returns $\langle\rangle$. However, the terms M_3 and M_3' are closed-argument bisimilar because the applications of V_3 and V_3' to the same closed argument will have the same behaviour; such arguments will not be able to throw an exception e. □

Example 4.2 *(Accumulation)* Let M_4 and M_4' of type $(1 \to 1) \times (1 \to (1 \to 1) \to \texttt{bool})$ be the terms:

$$M_4 \stackrel{\text{def}}{=} \texttt{new e. return}\,\langle \lambda\langle\rangle.\,\texttt{raise e},\ \lambda\langle\rangle.\,\texttt{return } V_4\rangle$$
$$M_4' \stackrel{\text{def}}{=} \texttt{new e. return}\,\langle \lambda\langle\rangle.\,\texttt{raise e},\ \lambda\langle\rangle.\,\texttt{return } V_4'\rangle$$

where

$$V_4 \stackrel{\text{def}}{=} \lambda \texttt{f. f}\,\langle\rangle\,\{\texttt{to x. return true},\ \ \texttt{catch e. return false}\}$$
$$V_4' \stackrel{\text{def}}{=} \lambda \texttt{f. f}\,\langle\rangle\,\{\texttt{to x. return true},\ \ \texttt{catch e. return true}\}$$

A distinguishing context is

$$C_4 \stackrel{\text{def}}{=} [\cdot]\ \texttt{to}\,\langle \texttt{f}, \texttt{g}\rangle.\ \texttt{g}\,\langle\rangle\ \texttt{to h. h f}$$

so that $C_4[M_4]$ returns `false` and $C_4[M_4']$ returns `true`. Note that the context C_4 employs accumulation: it uses the function `f`, obtained before the application of `g`, as an argument to `h`, obtained as a result of the application of `g`. The terms M_4 and M_4' are no-accumulation bisimilar, because by the time a non-accumulating context has obtained `h` by applying `g`, it has discarded the function `f`.

Because a no-accumulation bisimulation uses resourceful arguments, to prove the above terms no-accumulation bisimilar, we need the following fact: for any $\Delta; \texttt{y} : (1 \to 1) \to \texttt{bool} \vdash M : B$, where Δ does not contain e, if $\Delta \cdot \texttt{e}, M[V_4/\texttt{y}]$ evaluates to $\Theta \cdot \texttt{e}, W$ then there exists $\Theta; \texttt{y} : (1 \to 1) \to \texttt{bool} \vdash V : B$ such that

$W = V[V_4/\mathtt{y}]$ and $\Delta \cdot \mathtt{e}, M[V_4'/\mathtt{y}]$ evaluates to $\Theta \cdot \mathtt{e}, V[V_4'/\mathtt{y}]$, and likewise if it raises an exception. This is proved by induction on the big-step relation. □

4.2 Names

We add to λ the facility to generate fresh names that are values and may be compared for equality, similar to the nu-calculus [25,28]. Our syntax becomes

$$\begin{aligned} A ::= &\ \cdots \mid \mathtt{name} \\ V ::= &\ \cdots \mid \mathtt{m} \\ M ::= &\ \cdots \mid \mathtt{new\ x}.\ M \mid V = V \end{aligned}$$

Unlike previous work on environmental bisimulation for languages with first-class names (e.g. [31,15,5]) here we do not relate names in the relatees. Instead, private names that are revealed by related functions to their context are renamed into identical public names by means of ultimate pattern-matching. For example, the value

$$\begin{aligned} &\mathtt{m}_0, \mathtt{m}_1 \circledast \mathtt{n}_0, \mathtt{n}_1, \mathtt{n}_2; \vdash^{\mathtt{v}} \\ &\quad \langle\langle \mathtt{n}_2, \mathtt{m}_0\rangle, \langle \mathtt{n}_1, \langle \mathtt{n}_2, \mathtt{inl}\ \lambda \mathtt{x.return}\ \langle \mathtt{n}_0, \mathtt{n}_1\rangle\rangle\rangle\rangle \\ &\qquad : (\mathtt{name} \times \mathtt{name}) \times (\mathtt{name} \times (\mathtt{name} \times ((1 \to (\mathtt{name} \times \mathtt{name})) + 1))) \end{aligned} \tag{2}$$

has ultimate pattern

$$p \stackrel{\text{def}}{=} \langle\langle \mathtt{m}_2, \mathtt{m}_0\rangle, \langle \mathtt{m}_3, \langle \mathtt{m}_2, \mathtt{inl}\ -\rangle\rangle\rangle \tag{3}$$

To obtain (3) algorithmically, we scan (2) from left to right, converting a private name encountered for the first time into a public one, replacing functions by $-$ and retaining public names and tags. We encounter $\mathtt{n}_2$ before $\mathtt{n}_1$, therefore they are converted to $\mathtt{m}_2$ and $\mathtt{m}_3$ respectively.

We recover (2) from (3) by providing

- the filling of the hole under the new name scheme, viz. $\lambda \mathtt{x.return}\ \langle \mathtt{n}_0, \mathtt{m}_3\rangle$
- the list of converted private names in order of appearance, viz. $\mathtt{n}_2, \mathtt{n}_1$.

Once we have reformulated Thm. 3.1 along these lines, the notion of environmental bisimulation—and its two defective variants—is defined as in Sect. 3.2.

Example 4.3 *(Resourceful arguments)* Let M_5 and M_5' of type $(\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool}$ be

$$M_5 \stackrel{\text{def}}{=} \mathtt{new\,n.\,return}\ V_5 \qquad V_5 \stackrel{\text{def}}{=} \lambda \mathtt{f.\,f\,n}$$
$$M_5' \stackrel{\text{def}}{=} \mathtt{return}\ V_5' \qquad V_5' \stackrel{\text{def}}{=} \lambda \mathtt{f.\,new\,n.\,f\,n}$$

These are distinguished by

$$C_5 \stackrel{\text{def}}{=} [\cdot]\ \mathtt{to\ g.\ g}\,(\lambda \mathtt{n.\,g}\,(\lambda \mathtt{m.\,m} = \mathtt{n}))$$

Evidently $C_5[M_5]$ evaluates to `true` while $C_5[M_5']$ evaluates to `false`. On the other hand, M_5 and M_5' are closed-argument bisimilar, because a closed argument cannot know about `n`. In particular, it cannot store `n` when applied to it, for future use, because the language does not allow storage of names. □

Example 4.4 *(Accumulation)* The following are terms of type $1 \to \texttt{name}$:

$$M_6 \stackrel{\text{def}}{=} \texttt{new n. return } V_6 \qquad V_6 \stackrel{\text{def}}{=} \lambda\langle\rangle.\,\texttt{return n}$$
$$M_6' \stackrel{\text{def}}{=} \texttt{return } V_6' \qquad V_6' \stackrel{\text{def}}{=} \lambda\langle\rangle.\,\texttt{new n. return n}$$

These terms are only distinguished by a context that accumulates the function in its hole and applies it twice, such as the following:

$$C_6 \stackrel{\text{def}}{=} [\cdot]\ \texttt{to g. g}\,\langle\rangle\ \texttt{to x. g}\,\langle\rangle\ \texttt{to y. x} = \texttt{y}$$

$C_6[M_6]$ returns `true` but $C_6[M_6']$ returns `false`. However M_6 and M_6' are no-accumulation bisimilar since when applied just once (even to resourceful arguments) they behave the same. □

4.3 Polymorphism

We add polymorphism to λ, so our syntax of types and terms becomes

$$\begin{aligned} A ::= &\ \cdots \mid \textstyle\prod \texttt{X}.A \mid \sum \texttt{X}.A \\ V ::= &\ \cdots \mid \Lambda\texttt{X}.\,M \mid \texttt{rec f}\,\Lambda\texttt{X}.\,M \mid \langle A, V\rangle \\ M ::= &\ \cdots \mid V\,A \mid \texttt{match } V \texttt{ as } \langle \texttt{X}, \texttt{x}\rangle.\ M \end{aligned}$$

Using the ultimate pattern-matching theorem developed in [19], we again formulate environmental bisimulation incorporating the principles of resourceful arguments and accumulation, and the two defective versions. The following examples show the necessity of these principles.

Example 4.5 *(Resourceful arguments)* Consider the following existential packages of type $\sum \texttt{X}.\,(\texttt{X} \to (1 + \texttt{X})) \to \texttt{bool}$:

$$M_7 \stackrel{\text{def}}{=} \texttt{return } \langle 1, V_7\rangle$$
$$M_7' \stackrel{\text{def}}{=} \texttt{return } \langle \texttt{bool}, V_7'\rangle$$

where

$$V_7 \stackrel{\text{def}}{=} \lambda \mathtt{f}.\, \mathtt{f}\,\langle\rangle \text{ to } \{\mathtt{inl}\,\langle\rangle.\ \mathtt{return}\ \mathtt{false},\ \mathtt{inr}\,\langle\rangle.\ \mathtt{f}\,\langle\rangle;\ \mathtt{return}\ \mathtt{true}\}$$

$$V_7' \stackrel{\text{def}}{=} \lambda \mathtt{f}.\, \mathtt{f}\,\mathtt{false} \text{ to } \{\mathtt{inl}\,\langle\rangle.\ \mathtt{return}\ \mathtt{false}\ \ \mathtt{inr}\,\mathtt{true}.\ \mathtt{return}\ \mathtt{false}\ \ \mathtt{inr}\,\mathtt{false}.\ \mathtt{f}\,\mathtt{true}\ \mathtt{as}\ \{\mathtt{inl}\,\langle\rangle.\ \mathtt{return}\ \mathtt{true}\ \ \mathtt{inr}\,\mathtt{true}.\ \mathtt{return}\ \mathtt{true}\ \ \mathtt{inr}\,\mathtt{false}.\ \mathtt{return}\ \mathtt{false}\}\}$$

These are distinguished by the context

$$C_7 \stackrel{\text{def}}{=} [\cdot]\ \mathtt{to}\ \langle \mathtt{X}, \mathtt{g}\rangle.\ \mathtt{g}\,(\lambda \mathtt{y}{:}\mathtt{X}.\, \mathtt{g}\,(\lambda \mathtt{z}{:}\mathtt{X}.\, \mathtt{return}\ \mathtt{inr}\,\mathtt{y})\ \mathtt{to}\ \{\mathtt{true}.\ \mathtt{return}\ \mathtt{inr}\,\mathtt{y},\ \mathtt{false}.\ \mathtt{return}\ \mathtt{inl}\,\langle\rangle\})$$

The term $C_7[M_7] \Downarrow \mathtt{true}$, while $C_7[M_7'] \Downarrow \mathtt{false}$. But M_7 and M_7' are closed-argument bisimilar, because a closed argument for V_7 and V_7' has polymorphic type $\mathtt{X} \to (1 + \mathtt{X})$, and there are only three such: the constant functions $\lambda \mathtt{x}.\, \mathtt{return}\ \mathtt{inl}\,\langle\rangle$ and $\lambda \mathtt{x}.\, \mathtt{diverge}$, and the function $\lambda \mathtt{x}.\, \mathtt{return}\ \mathtt{inr}\,\mathtt{x}$. Evidently V_7 and V_7' behave the same way when applied to these arguments. □

Note that functions with existentially quantified argument type $\mathtt{X}$ (such as those in the following example) will necessarily need to be provided with resourceful arguments, because the context cannot construct closed values of type $\mathtt{X}$. Thus one may think that, for this language, a weaker notion of resourcefulness might be sound. However, the preceding example demonstrates the need not just for a resourceful argument but for one in which a resource (value from the inventory) is used under λ, just as in the rest of the languages we studied so far.

Example 4.6 *(Accumulation)* Accumulation is necessary since the context may receive new arguments to an old function. For example consider the terms M_8 and M_8' of type $\sum \mathtt{X}.\, (1 \to \mathtt{X}) \times (\mathtt{X} \to \mathtt{bool})$:

$$M_8 \stackrel{\text{def}}{=} \mathtt{return}\ \langle 1, V_8\rangle \qquad V_8 \stackrel{\text{def}}{=} \langle \lambda\langle\rangle.\, \mathtt{return}\ \langle\rangle, \lambda \mathtt{x}.\, \mathtt{return}\ \mathtt{true}\rangle$$

$$M_8' \stackrel{\text{def}}{=} \mathtt{return}\ \langle 1, V_8'\rangle \qquad V_8 \stackrel{\text{def}}{=} \langle \lambda\langle\rangle.\, \mathtt{return}\ \langle\rangle, \lambda \mathtt{x}.\, \mathtt{return}\ \mathtt{false}\rangle$$

These are distinguished by the context

$$C_8 \stackrel{\text{def}}{=} [\cdot]\ \mathtt{to}\ \langle \mathtt{X}, \langle \mathtt{f}, \mathtt{g}\rangle\rangle.\ \mathtt{f}\,\langle\rangle\ \mathtt{to}\ \mathtt{x}.\ \mathtt{g}\,\mathtt{x}$$

Evidently $C_8[M_8]$ returns $\mathtt{true}$ while $C_8[M_8']$ returns $\mathtt{false}$. However M_8 and M_8' are no-accumulation bisimilar because when the context receives the two functions

it cannot distinguish the right-hand function since it does not yet have a value to apply them to. □

5 Repeated Generation of Fresh Names

Another questionable point in the definition of environmental bisimulation is the ability of the context to add public names at each step of the bisimulation. Perhaps we could fix Δ_{pub} in Def. 3.2, giving a notion of Δ_{pub}-*bisimulation*, and then require terms $;\vdash M, M' : A$ to be Δ_{pub}-bisimilar for all Δ_{pub}? For the deterministic languages we considered so far, we believe this to be a sound modification.

In the presence of nondeterminism, however, there is no single answer; it depends on the kind of nondeterminism and on the contextual equivalence we consider. Here we study the extensions of the language with names from Sec. 4.2 with finite and countable nondeterminism. We believe the above restriction is sound for may-testing and, in the finitely non-deterministic setting, also for must-testing. Here we give two examples that show this to be unsound for must-testing, in the presence of countable nondeterminism, and for lower bisimilarity, in the presence of even finite nondeterminism.

In this section we abbreviate $\mathtt{nat} = \mathtt{rec\ X.}\ 1 + \mathtt{X}$ and $\mathtt{namelist} = \mathtt{rec\ x.}\ 1 + \mathtt{name} \times \mathtt{X}$. We use the usual constructors `zero` and `succ` for `nat`, and `empty` and `cons` for `namelist`. We use the following functions:

- $;\vdash^{\mathsf{v}} \mathtt{member} : (\mathtt{name} \times \mathtt{namelist}) \to \mathtt{bool}$ tells us whether its first argument appears in its second
- $;\vdash^{\mathsf{v}} \mathtt{distnames} : \mathtt{namelist} \to \mathtt{nat}$ returns the number of distinct names in its argument
- $;\vdash^{\mathsf{v}} \mathtt{mkfreshlist} : \mathtt{nat} \to \mathtt{namelist}$ creates a list of fresh names of length equal to its argument
- $;\vdash^{\mathsf{v}} \mathtt{min} : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$ is the minimum function.

5.1 Countable Nondeterminism and Must-Testing

We extend the language from Sec. 4.2 with both binary and countable (erratic) nondeterministic choice—of course the former is redundant. Our syntax is now

$$M ::= \quad \cdots \mid M\ \mathtt{or}\ M \mid \mathtt{choose\ x.}\ M$$

where `x` has type `nat`. The bigstep semantics is given in the standard manner [23]. First, we inductively define a relation $\Delta, M \Downarrow_A \Theta, V$, meaning that Δ, M may evaluate to Θ, V. Then, we coinductively define a predicate $\Delta, M \Uparrow_A$, meaning that Δ, M may diverge.

To soundly reason about both may-testing (possibility of convergence) and must-testing (impossibility of divergence) a set of relatees $\mathcal{R}$ needs to be a *convex* environmental bisimulation, i.e. if $\mathcal{C}$ and $\mathcal{C}'$ are $\mathcal{R}$-related configurations, then any convergence step or divergence that one may perform can be imitated by the other.

If we do not require divergence to be imitated, then $\mathcal{R}$ is merely a *lower* environmental bisimulation [34,20], which is sound for may-testing only.

Example 5.1 *(Generation)* We consider the terms M_9 and M_9' of type $T \stackrel{\text{def}}{=} \texttt{rec X. name} \to 1 + \texttt{X}$:

$$M_9 \stackrel{\text{def}}{=} \texttt{choose w. } V_9\, \langle \texttt{w}, \texttt{empty} \rangle$$
$$M_9' \stackrel{\text{def}}{=} M_9 \texttt{ or } V_9' \texttt{ empty}$$

where

$$\begin{aligned} V_9 \stackrel{\text{def}}{=}\ & \texttt{rec } (\texttt{f} : (\texttt{nat} \times \texttt{namelist}) \to T)\ \lambda(\texttt{c}, \texttt{lst}).\ \texttt{return fold } \lambda\texttt{n}. \\ & \quad \texttt{member } \langle \texttt{n}, \texttt{lst} \rangle \texttt{ to } \{ \\ & \quad\quad \texttt{true. return inl } \langle\rangle, \\ & \quad\quad \texttt{false. match c as } \{ \\ & \quad\quad\quad \texttt{zero. return inl } \langle\rangle, \\ & \quad\quad\quad \texttt{succ y. f } \langle \texttt{y}, \texttt{cons } \langle \texttt{n}, \texttt{lst} \rangle \rangle \texttt{ to g. return inr g } \} \ \} \\ V_9' \stackrel{\text{def}}{=}\ & \texttt{rec } (\texttt{f} : \texttt{namelist} \to T)\ \lambda\texttt{lst. return } \lambda\texttt{n}. \\ & \quad \texttt{member } \langle \texttt{n}, \texttt{lst} \rangle \texttt{ to } \{ \\ & \quad\quad \texttt{true. return inl } \langle\rangle, \\ & \quad\quad \texttt{false. f } (\texttt{cons } \langle \texttt{n}, \texttt{lst} \rangle) \texttt{ to g. return inr g } \} \end{aligned}$$

Here M_9 and M_9' return a "hungry function" that after accepting a name it returns another function that can accept more names, if the name is undoubtedly new to the function, and $\texttt{inl } \langle\rangle$, otherwise. The term M_9 grants the context an arbitrary finite number $\texttt{w}$ of distinct names it can remember; M_9' has the additional possibility of granting an unbounded memory.

The two terms are distinguished by the context

$$\begin{aligned} C_9 \stackrel{\text{def}}{=}\ & [\cdot] \texttt{ to f.} \\ & \quad (\texttt{rec } (\texttt{h} : T \to 1)\ \lambda(\texttt{fold y}).\ \texttt{new n. y n to } \{ \\ & \quad\quad \texttt{inl } \langle\rangle. \texttt{ return } \langle\rangle, \\ & \quad\quad \texttt{inr g. h g} \\ & \quad \})\, \texttt{f} \end{aligned}$$

Evidently $C_9[M_9']$ is able to diverge, whereas $C_9[M_9]$ is not. But for any fixed list of names Δ_{pub}, it is clear that M_9 and M_9' are convex Δ_{pub}-bisimilar. This example shows that, in order for environmental bisimulation to be sound for must-testing in the presence of countable nondeterminism, we cannot dispense with the provision for the context to generate fresh names at each step. □

5.2 Lower Bisimulation as a Congruence

Example 5.2 *(Generation)* Let M_{10} and M'_{10} be the following terms of type $\texttt{namelist} \to \texttt{nat}$:

$$M_{10} \stackrel{\text{def}}{=} \texttt{choose w. return } V_{10}$$
$$M'_{10} \stackrel{\text{def}}{=} \texttt{return } V'_{10} \texttt{ or } M_{10}$$

where

$$V_{10} \stackrel{\text{def}}{=} \lambda\texttt{lst. min}\langle\texttt{distnames(lst)}, \texttt{w}\rangle$$
$$V'_{10} \stackrel{\text{def}}{=} \lambda\texttt{lst. distnames(lst)}$$

The term M_{10} picks a number $\texttt{w}$ (or diverges) and returns a function that can count up to $\texttt{w}$ distinct names in a list; M'_{10} has the additional possibility of returning a function that can count an unbounded number of distinct names in a list.

Let C_{10} be the following context of type $1 \to \texttt{nat}$:

$$C_{10} \stackrel{\text{def}}{=} [\cdot] \texttt{ to f.}$$
$$\texttt{return } \lambda\langle\rangle.\ \texttt{choose k. mkfreshlist(k) to lst. f lst}$$

Now $C_{10}[M'_{10}]$ may return a function that, when applied to $\langle\rangle$, may return any natural number, whereas $C_{10}[M_{10}]$ cannot do this. Therefore any conceivable notion of lower bisimulation (or even lower simulation) must distinguish $C_{10}[M'_{10}]$ from $C_{10}[M_{10}]$, and hence, if it is a congruence, M'_{10} from M_{10}. Yet for every fixed list of names Δ_{pub}, M'_{10} and M_{10} are lower (in fact, even convex) Δ_{pub} bisimilar.

This example shows that, in order for lower environmental bisimulation to be a congruence (or simulation to be a precongruence) in the presence of nondeterminism, we cannot dispense with the provision for the context to generate fresh names at each step.

Note that for this example, we may replace countable nondeterministic choice by a variant that may also diverge, which is expressible using only binary nondeterministic choice ($\texttt{or}$) and recursion. The change is immaterial for lower bisimulation. □

6 Conclusions

In early developments of Environmental Bisimulation [31,32] it was realized that when we attempt to prove a contextual equivalence by brute force, we repeatedly find that the proof divides into two parts. One part is "boilerplate"; it does not change from example to example. The other part requires understanding of the particular example. Environmental bisimulation is simply a convenient way of packaging the boilerplate, so that only the example-specific part of the proof remains to be done. Once environmental relations are defined, the bisimulation conditions can be discovered by the soundness proof of these relations [12].

This paper motivates in retrospect the necessity of the complexity in the conditions of environmental bisimulation by giving a collection of examples in a variety of higher-order languages. The examples show, for these languages, the need to deviate in two significant ways from the standard notion of applicative bisimulation.

There are also other sound operational techniques for these languages which resemble applicative bisimulation even less. Notably open bisimulation [19,18,26,9] and complete traces [17]; and logical relations in conjunction with step-indexing [3,2,6] and biorthogonal closure [24]. A logical relation is typically a single relation relating all contextually equivalent terms—thus accumulation is a given. It also supplies related arguments to related functions which, because the logical relation is a congruence, follows the principle of resourceful arguments. Open bisimulation and complete traces for languages similar to those studied here accumulate values [17,9]. However, they follow a quite different approach for functions: they provide them with fresh identifiers, and make the applications of identifiers observable.

Jeffrey and Rathke [10] showed the unsoundness of an accumulating but closed-argument form of bisimulation for the nu-calculus. The addition of infinitely many global name references makes their bisimulation sound and complete. That language is unaffected by Ex. 4.3 because names may be stored, and unaffected by Ex. 3.4 because there are no local references. On the other hand, their notion of bisimulation for a fragment of Concurrent ML [11] both accumulates values and uses resourceful arguments.

The notion of environmental bisimulation in the polymorphic setting introduced in [32] allows the context to supply resourceful type arguments to polymorphic functions. It would be interesting to know whether this is necessary, in the light of genericity results such as those in [21].

References

[1] Abramsky, S., *The lazy lambda calculus*, in: D. A. Turner, editor, *Research Topics in Functional Programming*, Addison-Wesley, Boston, MA, USA, 1990 pp. 65–116.

[2] Ahmed, A., A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan and D. C. Wang, *Semantic foundations for typed assembly languages*, ACM Trans. Prog. Lang. Syst. **32** (2010), pp. 1–67.

[3] Appel, A. W. and D. McAllester, *An indexed model of recursive types for foundational proof-carrying code*, ACM Trans. Prog. Lang. Sys. **23** (2001), pp. 657–683.

[4] Benton, N. and A. Kennedy, *Exceptional syntax*, J. Funct. Program. **11** (2001), pp. 395–410.

[5] Benton, N. and V. Koutavas, *A mechanized bisimulation for the nu-calculus*, Technical Report MSR-TR-2008-129, Microsoft Research (2008).

[6] Dreyer, D., G. Neis and L. Birkedal, *The impact of higher-order state and control effects on local relational reasoning*, in: *ACM SIGPLAN International Conference on Functional Programming* (2010), pp. 143–156.

[7] Gordon, A. D. and G. D. Rees, *Bisimilarity for a first-order calculus of objects with subtyping*, in: *ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (1996), pp. 386–395.

[8] Howe, D. J., *Proving congruence of bisimulation in functional programming languages*, Information and Computation **124** (1996), pp. 103–112.

[9] Jagadeesan, R., C. Pitcher and J. Riely, *Open bisimulation for aspects*, in: *International Conference on Aspect-Oriented Software Development* (2007).

[10] Jeffrey, A. and J. Rathke, *Towards a theory of bisimulation for local names*, in: *IEEE Symposium on Logic in Computer Science* (1999).

[11] Jeffrey, A. and J. Rathke, *A theory of bisimulation for a fragment of Concurrent ML with local names*, in: *IEEE Symposium on Logic in Computer Science, 2000*, 2000, pp. 311 –321.

[12] Koutavas, V., "Reasoning about Imperative and Higher-Order Programs," Ph.D. thesis, Northeastern University (2008).

[13] Koutavas, V., P. B. Levy and E. Sumii, *Limitations of applicative bisimulation (preliminary report)*, in: A. Ahmed, N. Benton, L. Birkedal and M. Hofmann, editors, *Modelling, Controlling and Reasoning About State*, number 10351 in Dagstuhl Seminar Proceedings (2010).
URL http://drops.dagstuhl.de/opus/volltexte/2010/2807

[14] Koutavas, V. and M. Wand, *Bisimulations for untyped imperative objects*, in: P. Sestoft, editor, *European Symposium on Programming*, LNCS **3924** (2006), pp. 146–161.

[15] Koutavas, V. and M. Wand, *Small bisimulations for reasoning about higher-order imperative programs*, in: *ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages* (2006), pp. 141–152.

[16] Koutavas, V. and M. Wand, *Reasoning about class behavior*, FOOL/WOOD Workshop (2007).

[17] Laird, J., *A fully abstract trace semantics for general references*, in: L. Arge, C. Cachin, T. Jurdzinski and A. Tarlecki, editors, *International Colloquium on Automata, Languages and Programming*, LNCS **4596** (2007), pp. 667–679.

[18] Lassen, S. B. and P. B. Levy, *Typed normal form bisimulation*, in: J. Duparc and T. Henzinger, editors, *Computer Science and Logic*, LNCS **4646**, 2007.

[19] Lassen, S. B. and P. B. Levy, *Typed normal form bisimulation for parametric polymorphism*, in: *LICS* (2008), pp. 341–352.

[20] Lassen, S. B. and C. S. Pitcher, *Similarity and bisimilarity for countable non-determinism and higher-order functions (extended abstract)*, Electronic Notes in Theoretical Computer Science **10** (1998), pp. 246–266.

[21] Longo, G., K. Milsted and S. Soloviev, *The genericity theorem and the notion of parametricity in the polymorphic λ-calculus*, in: *IEEE Logic in Computer Science, 1993. LICS '93., Proceedings of Eighth Annual IEEE Symposium on*, 1993, pp. 6 –14.

[22] Mason, I. A. and C. L. Talcott, *Equivalence in functional languages with effects*, Journal of Functional Programming **1** (1991), pp. 287–327.

[23] Moran, A. K., *Natural semantics for non-determinism*, Licentiate Thesis, Chalmers University of Technology and University of Göteborg, Sweden (1994).

[24] Pitts, A. and I. Stark, *Operational reasoning for functions with local state*, in: A. Gordon and A. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, Cambridge University Press, 1998 pp. 227–273.

[25] Pitts, A. M. and I. D. B. Stark, *Observable properties of higher order functions that dynamically create local names, or: What's new?*, in: *Mathematical Foundations of Computer Science*, LNCS **711** (1993), pp. 122–141.

[26] Sangiorgi, D., *The lazy lambda calculus in a concurrency scenario*, Information and Computation **111** (1994), pp. 120–153.

[27] Sangiorgi, D., N. Kobayashi and E. Sumii, *Environmental bisimulations for higher-order languages*, in: *IEEE Symposium on Logic in Computer Science* (2007), pp. 293–302.

[28] Stark, I. D. B., "Names and Higher-Order Functions," Ph.D. thesis, University of Cambridge, Cambridge, UK (1994), Technical Report 363.

[29] Sumii, E. and B. C. Pierce, *A bisimulation for dynamic sealing*, in: *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (2004), pp. 161–172.

[30] Sumii, E. and B. C. Pierce, *A bisimulation for type abstraction and recursion*, in: *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (2005), pp. 63–74.

[31] Sumii, E. and B. C. Pierce, *A bisimulation for dynamic sealing*, Theoretical Computer Science **375** (2007), pp. 169–192, (extended abstract appeared in [29]).

[32] Sumii, E. and B. C. Pierce, *A bisimulation for type abstraction and recursion*, Journal of the ACM **54** (2007), pp. 1–43, (extended abstract appeared in [30]).

[33] Tiuryn, J. and M. Wand, *Untyped lambda-calculus with input-output*, in: H. Kirchner, editor, *Colloquium on Trees in Algebra and Programming*, LNCS **1059** (1996), pp. 317–329.

[34] Ulidowski, I., *Equivalences on observable processes*, in: A. Scedrov, editor, *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science* (1992), pp. 148–161.