Towards a theory of bisimulation for local names

Alan Jeffrey
CTI, DePaul University
243 South Wabash Ave
Chicago IL 60604, USA
ajeffrey@cs.depaul.edu

Julian Rathke
COGS, University of Sussex
Brighton BN1 9QH, UK
julianr@cogs.susx.ac.uk

Extended abstract submitted to LICS 99

Abstract

Pitts and Stark have proposed the ν-calculus as a language for investigating the interaction of unique name generation and higher-order functions. They developed a sound model based on logical relations, but left completeness as an open problem. In this paper, we develop a complete model based on bisimulation for a labelled transition system semantics. We show that bisimulation is complete, but not sound, for the ν-calculus. We also show that by adding assignment to the ν-calculus, bisimulation becomes sound and complete. The analysis used to obtain this result illuminates the difficulties involved in finding fully abstract models for ν-calculus proper.

Keywords: semantics, bisimulation, nominal calculi.

1 Introduction

The use of localised information in computing is endemic. In a workplace, where the prevalence of networks and mobile agents is increasing, the concepts of private and public data are of great importance, and the question of how secrecy can be maintained is a significant one. Issues of privacy are important for notions of locally defined names such as communication channels [11], references [16], encryption keys [4], or locations [5]. Gordon has described such languages as *nominal calculi* [7].

To this end, Pitts and Stark introduced the ν-calculus [14, 15, 21] as a minimal higher-order nominal language, by extending the simply typed λ-calculus with an abstract type of names, together with a name generator and an equality test. Even this small language allows one to model and reason about visibility and scoping quite thoroughly. In fact, the interaction between locally declared names and higher-order functions is extremely complex, and at present there is no known fully abstract model of ν-calculus beyond first-order types.

In this paper we revisit the ν-calculus of Pitts and Stark and provide a novel treatment of the semantics of local names using a labelled transition systems (lts). In order for the standard notion of weak bisimulation to be fully abstract for contextual equivalence, we have:

- For *completeness*, we show that each transition $\stackrel{\gamma}{\Longrightarrow}$ corresponds to a small piece of context $\mathcal{C}_\gamma[t]$ such that $t \stackrel{\gamma}{\Longrightarrow} v$ iff $\mathcal{C}_\gamma[t] \Rightarrow (v, \texttt{true})$. (We call such lts's *contextual*: the notion that transition labels should correspond to small contexts appears to be folklore, and has only recently been investigated formally by Sewell [20].)
- For *soundness*, we show that bisimulation is a congruence.

We notice that our approach to characterising contextual equivalence is already in sharp contrast to Pitts and Stark. They propose logical relations as an operational proof technique for establishing contextual equivalence of ν-calculus terms. The logical relation can easily be construed as a form of bisimulation on an lts, but the labels which would have to be used are not contextual—this compromises completeness in order to obtain a direct proof of soundness for their technique.

In the case of the λ-calculus, we revisit Gordon [6] and Bernstein and Stark's [1] presentation of an lts semantics of the λ-calculus. Completeness is routine, and soundness follows by using Howe's [10] technique to show bisimulation to be a congruence.

For the ν-calculus, there is a simple extension of the lts for the λ-calculus to give a semantics for local names, but it transpires that bisimulation fails to be a congruence. We make explicit the reason for this failure and argue that the problem arises due to the paucity of contexts in ν-calculus and that, by extending the base calculus with more realistic features, the problem dissolves. By 'more realistic features' we mean any side-effecting operators which have the capability to model leaking secrets.

The particular extension we select for study in this paper is the νref-calculus, given by adding global references which store names. Leaking a secret name is implemented by assigning it to a shared reference. We consider this to be a minimal extension of the language for which our proof techniques are successful. We demonstrate that bisimulation for the extended language is actually a congruence and thus achieve a full abstraction result for contextual equivalence.

We would like to thank Karen Bernstein, Matthew Hennessy, Guy McCusker and Ian Stark for discussions about this paper.

2 λ-calculus

In this section we consider a simply typed call-by-value lambda-calculus with booleans and pairing.

We present the usual small-step reduction semantics, and a slight modification of Gordon's [6] and Bernstein and Stark's [1] labelled transition system semantics. We then show that bisimulation (defined on the lts semantics) is both sound and complete for contextual equivalence (defined on the reduction semantics).

We show completeness (contextual equivalence implies bisimulation) by observing that each label of the lts semantics corresponds to a small piece of context, and so each lts transition can be matched by contextual equivalence. This is proof is inspired by Hennessy's [8] proofs for may and must testing.

Soundness (bisimulation implies contextual equivalence) follows immediately once we can show that bisimulation is a congruence. We briefly discuss Gordon's [6] presentation of Howe's [10] technique for this proof, concentrating on the aspect of the proof which is problematical for the ν-calculus.

2.1 Syntax and type rules

The grammar of types is given by:

$$\sigma ::= \mathtt{unit} \mid \mathtt{bool} \mid \sigma \times \sigma \mid \sigma \to \sigma$$

The grammar of values is given by:

$$v ::= x \mid () \mid \mathtt{true} \mid \mathtt{false} \mid (v,v) \mid \lambda x:\sigma\,.\,t$$

where x is drawn from some infinite set of variables, and the grammar of terms is given by:

$$t ::= v \mid \mathtt{if}\ v\ \mathtt{then}\ t\ \mathtt{else}\ t \mid \mathtt{fst}\ v \mid \mathtt{snd}\ v \mid vv \mid \mathtt{let}\ x\ =\ t\ \mathtt{in}\ t$$

We have taken the liberty of limiting much of the syntax to values: we can use obvious syntax sugar (in the spirit of Moggi's [13] computational monads) to extend these to terms. For example the term (t,t') is defined:

$$(t,t') \stackrel{\mathsf{def}}{=} \mathtt{let}\ x\ =\ t\ \mathtt{in}\ \mathtt{let}\ x'\ =\ t'\ \mathtt{in}\ (x,x')$$

The type system for this λ-calculus is standard, and is given by judgements of the form $\Gamma \vdash t : \sigma$ where Γ is a type environment of the form $x_1:\sigma_1,\ldots,x_n:\sigma_n$ for disjoint $x_1,\ldots,x_n$.

2.2 Reduction semantics

We now consider the operational semantics of the λ-calculus, given as *small-step* reductions $t \xrightarrow{\tau} t'$. First, we give the atomic rules for reduction:

$$\begin{aligned}
\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ t\ \mathtt{else}\ t' &\xrightarrow{\tau} t\\
\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ t\ \mathtt{else}\ t' &\xrightarrow{\tau} t'\\
\mathtt{fst}\ (v,v') &\xrightarrow{\tau} v\\
\mathtt{snd}\ (v,v') &\xrightarrow{\tau} v'\\
(\lambda x:\sigma\,.\,t)v &\xrightarrow{\tau} t[v/x]\\
\mathtt{let}\ x\ =\ v\ \mathtt{in}\ t &\xrightarrow{\tau} t[v/x]
\end{aligned}$$

then, following Wright and Felleisen [22] we allow reduction in any *evaluation context* given by the grammar:

$$\mathcal{E} ::= \cdot \mid \mathtt{let}\ x\ =\ \mathcal{E}\ \mathtt{in}\ t$$

with the rule:

$$\frac{t \xrightarrow{\tau} t'}{\mathcal{E}[t] \xrightarrow{\tau} \mathcal{E}[t']}$$

Let $\Rightarrow$ denote the reflexive, transitive closure of $\xrightarrow{\tau}$.

We can define the usual notion of contextual equivalence: two terms are considered equivalent whenever there is no boolean context which can distinguish between them: for terms $\Gamma \vdash t : \sigma$ and $\Gamma \vdash t' : \sigma$, we define $\Gamma \vdash t \approx_{ctx} t' : \sigma$ if for all closing contexts $\mathcal{C}$ of type `bool`, we have $\mathcal{C}[t] \Rightarrow \mathtt{true}$ iff $\mathcal{C}[t'] \Rightarrow \mathtt{true}$.

2.3 Labelled transition system semantics

We now introduce a lts semantics for λ-calculus terms.

Reductions are of the form $v \xrightarrow{\gamma} t$ where the grammar of labels is given by:

$$\gamma ::= \mathtt{true} \mid \mathtt{false} \mid @v \mid \mathtt{copy} \mid \mathtt{discard} \mid \mathtt{l}.\gamma \mid \mathtt{r}.\gamma$$

Atomic rules for reductions:

$$\begin{aligned}
\mathtt{true} &\xrightarrow{\mathtt{true}} ()\\
\mathtt{false} &\xrightarrow{\mathtt{false}} ()\\
\lambda x:\sigma\,.\,t &\xrightarrow{@v} t[v/x] \qquad (\text{where } \vdash v:\sigma)
\end{aligned}$$

diagonal and terminal transitions:

$$\begin{aligned}
v &\xrightarrow{\mathtt{copy}} (v,v)\\
v &\xrightarrow{\mathtt{discard}} ()
\end{aligned}$$

and inference rules for pair contexts:

$$\frac{v \xrightarrow{\gamma} t}{(v,v') \xrightarrow{\mathtt{l}.\gamma} (t,v')} \qquad \frac{v' \xrightarrow{\gamma} t'}{(v,v') \xrightarrow{\mathtt{r}.\gamma} (v,t')}$$

recalling that (t,v) and (v,t) can be defined as syntax sugar using let.

We can then include the reduction semantics to give reductions of the form $t \xrightarrow{\alpha} t'$ where:

$$\alpha ::= \tau \mid \gamma$$

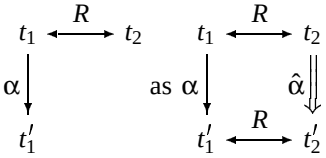
Define $t \stackrel{\gamma}{\Longrightarrow} t'$ as $t \Rightarrow v \xrightarrow{\gamma} t'$. Define $t \stackrel{\hat{\alpha}}{\Longrightarrow} t'$ as $t \Rightarrow t'$ when α is τ and $t \stackrel{\alpha}{\Longrightarrow} t'$ otherwise.

A *type-indexed relation* on terms R is a family of relations $R_{\Gamma,\sigma}$ such that if $t\ R_{\Gamma,\sigma}\ t'$ then $\Gamma \vdash t : \sigma$ and $\Gamma \vdash t' : \sigma$. We shall usually write $\Gamma \vDash t\ R\ t' : \sigma$ for $t\ R_{\Gamma,\sigma}\ t'$, and often elide the type information where it is obvious from context.

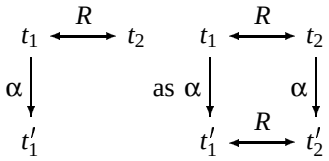
The *open extension* R° of a type-indexed relation on closed terms R is given by:

$$\frac{\forall(\vdash \overline{v}:\overline{\sigma})\,.\,(\vDash t[\overline{v}/\overline{x}]\ R\ t'[\overline{v}/\overline{x}] : \sigma)}{\overline{x}:\overline{\sigma} \vDash t\ R^\circ\ t' : \sigma}$$

A *simulation* is a type-indexed relation on closed terms R such that the following diagram can be completed:



A *strong simulation* is a type-indexed relation on closed terms R such that the following diagram can be completed:



A (strong) bisimulation is a (strong) simulation whose inverse is a simulation.

Let $\approx$ be the largest bisimulation, and let $\sim$ be the largest strong bisimulation.

2.4 Example

Let *not* be defined:

$$not \stackrel{\text{def}}{=} \lambda x : \texttt{bool} \,.\, \texttt{if}\ x\ \texttt{then}\ \texttt{false}\ \texttt{else}\ \texttt{true}$$

then one sample reduction of *not* is:

$$\begin{array}{rcl} not & \stackrel{\texttt{copy}}{\Longrightarrow} & (not, not) \\ & \stackrel{\texttt{l.@true}}{\Longrightarrow} & (not(\texttt{true}), not) \\ & \stackrel{\tau}{\Longrightarrow} & (\texttt{false}, not) \\ & \stackrel{\texttt{r.@false}}{\Longrightarrow} & (\texttt{false}, not(\texttt{false})) \\ & \stackrel{\tau}{\Longrightarrow} & (\texttt{false}, \texttt{true}) \\ & \stackrel{\texttt{l.false}}{\Longrightarrow} & ((), \texttt{true}) \\ & \stackrel{\texttt{r.true}}{\Longrightarrow} & ((), ()) \end{array}$$

showing how *not* evaluates when applied to `true` or `false`.

2.5 Completeness

In this section, we shall show that bisimulation is *complete*, that is:

$$\text{if } t \approx_{ctx} t' \text{ then } t \approx^{\circ} t'$$

First we observe that the λ-calculus is deterministic and normalizing, and so bisimulation and trace equivalence coincide.

We then show that contextual equivalence implies trace equivalence by constructing a context $\mathcal{C}_{\overline{\gamma}}$ for each sequence of labels $\overline{\gamma}$ such that such that:

$$t \stackrel{\overline{\gamma}}{\Longrightarrow} v \quad \text{iff} \quad \mathcal{C}_{\overline{\gamma}}[t] \Rightarrow (v, \texttt{true})$$

For example:

$$\begin{array}{rcl} \mathcal{C}_{\texttt{copy}}[t] & \stackrel{\text{def}}{=} & \texttt{let}\ x\ =\ t\ \texttt{in}\ ((x,x),\texttt{true}) \\ \mathcal{C}_{\texttt{l}.\gamma}[t] & \stackrel{\text{def}}{=} & \texttt{let}\ (x_1,x_2)\ =\ t \\ & & \texttt{in let}\ (x_1',x_2')\ =\ \mathcal{C}_{\gamma}[x_1] \\ & & \quad \texttt{in}\ ((x_1',x_2),x_2') \\ \mathcal{C}_{\overline{\gamma},\overline{\gamma}'}[t] & \stackrel{\text{def}}{=} & \texttt{let}\ (x_1,x_2)\ =\ \mathcal{C}_{\overline{\gamma}}[t] \\ & & \texttt{in let}\ (x_1',x_2')\ =\ \mathcal{C}_{\overline{\gamma}'}[x_1] \\ & & \quad \texttt{in}\ (x_1',x_2 \wedge x_2') \end{array}$$

using some obvious syntax sugar on terms.

Theorem 2.1 (completeness for λ-calculus)
If $\Gamma \vDash t \approx_{ctx} t' : \sigma$ *then* $\Gamma \vDash t \approx^{\circ} t' : \sigma$.

2.6 Soundness

In this section, we shall show that bisimulation is *sound*, that is:

$$\text{if } t \approx^{\circ} t' \text{ then } t \approx_{ctx} t'$$

This result is immediate from the result that bisimulation is a congruence, for which we adopt Howe's technique [10], following Gordon [6].

For any type-indexed relation R, let $\widehat{R}$ be defined such that for each type rule in the language:

$$\frac{\overline{\Gamma} \vdash \overline{t} : \overline{\sigma}}{\Gamma \vdash op(\overline{t}) : \sigma}$$

we have:

$$\frac{\overline{\Gamma} \vDash \overline{t}\ R\ \overline{t}' : \overline{\sigma}}{\Gamma \vDash op(\overline{t})\ \widehat{R}\ op(\overline{t}') : \sigma}$$

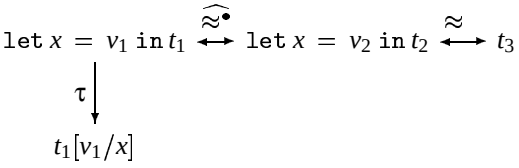
For any type-indexed relation R on closed terms, let $R^{\bullet}$ be defined:

$$\frac{t_1\ \widehat{R^{\bullet}}\ t_2\ R^{\circ}\ t_3}{t_1\ R^{\bullet}\ t_3}$$

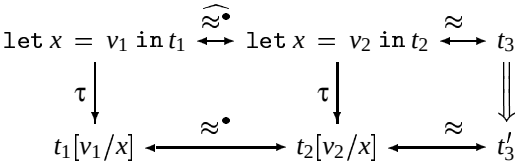
Howe's proof depends first on showing that $\approx^{\bullet}$ is *substitutive on values*:

$$\begin{array}{l} \text{if } t_1 \approx^{\bullet} t_2 \\ \text{and } v_1 \approx^{\bullet} v_2 \\ \text{then } t_1[v_1/x] \approx^{\bullet} t_2[v_2/x] \end{array}$$

and then showing that $\approx^{\bullet}$ is a bisimulation on closed terms. The only tricky case is let-β, where we complete:



as:



which commutes because $\approx^{\bullet}$ is substitutive on values.

From this, it is routine to show that bisimulation is a congruence, and so is sound.

Theorem 2.2 (soundness for λ-calculus)
If $\Gamma \vDash t \approx^{\circ} t' : \sigma$ *then* $\Gamma \vDash t \approx_{ctx} t' : \sigma$.

2.7 Comments

The astute reader will notice that the `copy` and `discard` transitions are redundant in this setting. In fact, it is a well known property of pure functional languages that 'operational extensionality' holds, that is, contextual equivalence can be verified by using applicative contexts alone. This does certainly not hold true of the extensions to the λ-calculus which we will consider later in this paper where operational extensionality fails.

In a similar vein, we notice that the use of `l.` and `r.` tags rather than Gordon's `fst` and `snd` transitions is also unnecessary here because pairing forms a product on values. In later sections, because of the presence of side-effects, the pairing operator is no longer a product, but is symmetric monoidal.

It is an important feature of the transition systems being used here, and also those of [6, 1] that they are *applicative* in nature. That is, any arbitrary pieces of code being carried in the label is always of lower order type than the term under scrutiny.

3 ν-calculus

We now extend the λ-calculus with unique name generation and equality testing, in order to investigate Pitts and Stark's [14] ν-calculus.

Pitts and Stark have demonstrated that finding a sound and complete semantics for the ν-calculus is a difficult open problem. They provide a sound (but incomplete) semantics using logical relations. In this section, we provide an 'upper bound' to complement their 'lower bound' by presenting a bisimulation which is complete (but unsound). We observe that our complete bisimulation provides a hands-on proof method for establishing contextual inequivalence and allows one to construct distinguishing contexts in a piecemeal fashion.

3.1 Syntax and type rules

Extend the grammar of types with:

$$\sigma ::= \cdots \mid \texttt{name}$$

Extend the grammar of values with:

$$v ::= \cdots \mid n$$

Extend the grammar of terms with:

$$t ::= \cdots \mid \nu n \,.\, t \mid v = v$$

Extend the type judgements $\Gamma \vdash t : \sigma$ to include a *name context* Δ of the form $n_1, ..., n_n$ for distinct n_i, so judgements are now of the form $\Gamma; \Delta \vdash t : \sigma$. The type rules for the new terms are:

$$\frac{}{\Gamma; \Delta, n, \Delta' \vdash n : \texttt{name}} \qquad \frac{\Gamma; \Delta, n \vdash t : \sigma}{\Gamma; \Delta \vdash \nu n \,.\, t : \sigma}$$

$$\frac{\Gamma; \Delta \vdash v : \texttt{name} \quad \Gamma; \Delta \vdash v' : \texttt{name}}{\Gamma; \Delta \vdash v = v' : \texttt{bool}}$$

The other rules do not change the name context.

3.2 Reduction semantics

Terms no longer reduce to values, instead they now reduce to *prevalues* of the form:

$$p ::= \nu \overline{n} \,.\, v$$

Extend the reduction relation with (when $n \neq n'$):

$$n = n \xrightarrow{\tau} \texttt{true}$$
$$n = n' \xrightarrow{\tau} \texttt{false}$$

Extend the grammar of evaluation contexts by:

$$\mathcal{E} ::= \cdots \mid \nu n \,.\, \mathcal{E}$$

Replace the let-β reduction rule by:

$$\texttt{let}\, x = \nu \overline{n} \,.\, v \,\texttt{in}\, t \xrightarrow{\tau} \nu \overline{n} \,.\, t[v/x]$$

where we α-convert $\nu \overline{n} \,.\, v$ if necessary to ensure that none of the free names in t are captured.

There is an obvious translation from Pitts and Stark's ν-calculus into ours (theirs does not include pairing), and it is routine to show that this translation is adequate.

The definition of contextual equivalence remains the same, except that the results of a test can include some private names: $t \approx_{ctx} t'$ whenever for all closing contexts $\mathcal{C}$ of type `bool`, we have $\mathcal{C}[t] \Rightarrow \nu \overline{n} \,.\, \texttt{true}$ iff $\mathcal{C}[t'] \Rightarrow \nu \overline{n}' \,.\, \texttt{true}$.

3.3 Labelled transition system semantics

We can no longer define the lts semantics as judgements $v \xrightarrow{\gamma} t$, for two reasons:

- Terms may reduce down to prevalues now, rather than values, so transitions should be of the form $p \xrightarrow{\gamma} t$.
- One of the allowed transitions allows a private name to become public, and we α-convert the name to ensure it does not clash with any existing public names. To do this, we carry an environment of existing public names, so transitions should be of the form $(\Delta \vdash p) \xrightarrow{\gamma} (\Delta, \Delta' \vdash t)$. Note that transitions can add new public names, but not remove any.

We extend the grammar of labels by:

$$\gamma ::= \cdots \mid n \mid \nu n \mid \texttt{weaken}\, n$$

These labels can be read as 'the term announces a public name', 'the term announces a private name, and makes it public', and 'the environment announces a new public name'.

A public name can be announced:

$$(\Delta \vdash n) \xrightarrow{n} (\Delta \vdash ())$$

The environment can invent new public names:

$$(\Delta \vdash p) \xrightarrow{\texttt{weaken}\, n} (\Delta, n \vdash p)$$

The context $\nu n \,.\, \cdot$ is a reduction context:

$$\frac{(\Delta, n \vdash p) \xrightarrow{\gamma} (\Delta, n, \Delta' \vdash t)}{(\Delta \vdash \nu n \,.\, p) \xrightarrow{\gamma} (\Delta, \Delta' \vdash \nu n \,.\, t)} \; [n \text{ not in } \gamma]$$

Private names can announce themselves and become public:

$$\frac{(\Delta, n \vdash p) \xrightarrow{\iota.n} (\Delta, n \vdash t)}{(\Delta \vdash \nu n \,.\, p) \xrightarrow{\iota.\nu n} (\Delta, n \vdash t)}$$

where $\iota.$ is a sequence of `l.` and `r.` tags. The side-condition on application is weakened to allow values to have free public names:

$$(\Delta \vdash \lambda x : \sigma \,.\, t) \xrightarrow{@v} (\Delta \vdash t[v/x]) \qquad (\text{where } \Delta \vdash v : \sigma)$$

We define the weak reduction $(\Delta \vdash t) \stackrel{\gamma}{\Longrightarrow} (\Delta' \vdash t')$ whenever $t \Rightarrow p$ and $(\Delta \vdash p) \stackrel{\gamma}{\rightarrow} (\Delta' \vdash t')$. We can then define bisimulation as usual between configurations $(\Delta \vdash t)$.

3.4 Example

Consider the terms:

$$\nu n \,.\, \lambda x : \mathtt{unit} \,.\, n \not\approx \lambda x : \mathtt{unit} \,.\, \nu n \,.\, n$$

These are not bisimilar because the first term has the reduction:

$$\begin{array}{rcl}
\nu n \,.\, \lambda x : \mathtt{unit} \,.\, n & \stackrel{\mathtt{copy}}{\Longrightarrow} & \nu n \,.\, (\lambda x : \mathtt{unit} \,.\, n, \lambda x : \mathtt{unit} \,.\, n) \\
& \stackrel{\mathtt{l}.@()}{\Longrightarrow} & \nu n \,.\, (n, \lambda x : \mathtt{unit} \,.\, n) \\
& \stackrel{\mathtt{r}.@()}{\Longrightarrow} & \nu n \,.\, (n, n) \\
& \stackrel{\mathtt{l}.\nu n}{\Longrightarrow} & ((), n) \\
& \stackrel{\mathtt{r}.n}{\Longrightarrow} & ((), ())
\end{array}$$

which the second term can only match:

$$\begin{array}{rcl}
\lambda x : \mathtt{unit} \,.\, \nu n \,.\, n & \stackrel{\mathtt{copy}}{\Longrightarrow} & (\lambda x : \mathtt{unit} \,.\, \nu n \,.\, n, \lambda x : \mathtt{unit} \,.\, \nu n \,.\, n) \\
& \stackrel{\mathtt{l}.@()}{\Longrightarrow} & \nu n \,.\, (n, \lambda x : \mathtt{unit} \,.\, \nu n \,.\, n) \\
& \stackrel{\mathtt{r}.@()}{\Longrightarrow} & \nu n \,.\, \nu n' \,.\, (n, n') \\
& \stackrel{\mathtt{l}.\nu n}{\Longrightarrow} & \nu n' \,.\, ((), n')
\end{array}$$

At this point the term cannot match the last $\stackrel{\mathtt{r}.n}{\Longrightarrow}$ transition performed by the first term because its only move is:

$$\nu n' \,.\, ((), n') \stackrel{\mathtt{r}.\nu n'}{\Longrightarrow} ((), ())$$

Note that this example relies crucially on the use of `copy`, $\mathtt{l}.\gamma$ and $\mathtt{r}.\gamma$ transitions.

3.5 Completeness

Completeness for the ν-calculus follows in the same way as it does for the λ-calculus. For any sequence $\overline{\gamma}$ we define a context $\mathcal{C}^{\Delta}_{\overline{\gamma}}$ such such that:

$$\begin{array}{l}
(\Delta \vdash t) \stackrel{\overline{\gamma}}{\Longrightarrow} (\Delta, \Delta' \vdash \nu \overline{n} \,.\, v) \\
\quad \text{iff} \quad (\Delta \vdash \mathcal{C}^{\Delta}_{\overline{\gamma}}[t]) \Rightarrow (\Delta \vdash \nu \overline{n} \,.\, (\Delta', v, \mathtt{true}))
\end{array}$$

The only interesting cases are:

$$\begin{array}{rcl}
\mathcal{C}^{\Delta}_{\nu n}[t] & \stackrel{\mathsf{def}}{=} & \mathtt{let}\, x \,=\, t \\
& & \mathtt{in}\, \mathtt{if}\, x \in \Delta \\
& & \quad \mathtt{then}\, (x, (), \mathtt{false}) \\
& & \quad \mathtt{else}\, (x, (), \mathtt{true}) \\
\mathcal{C}^{\Delta}_{\overline{\gamma},\overline{\gamma}}[t] & \stackrel{\mathsf{def}}{=} & \mathtt{let}\, (\overline{x}, x_1, x_2) \,=\, \mathcal{C}^{\Delta}_{\overline{\gamma}}[t] \\
& & \mathtt{in}\, \mathtt{let}\, (\overline{x}', x_1', x_2') \,=\, \mathcal{C}^{\Delta,\overline{x}}_{\overline{\gamma}}[x_1] \\
& & \quad \mathtt{in}\, (\overline{x}, \overline{x}', x_1', x_2 \wedge x_2')
\end{array}$$

using some obvious syntax sugar for terms. The result then follows as for the λ-calculus.

Theorem 3.1 (completeness for ν-calculus)
If $\Gamma; \Delta \vDash t \approx_{ctx} t' : \sigma$ *then* $\Gamma; \Delta \vDash t \approx^{\circ} t' : \sigma$.

3.6 Lack of soundness

Unfortunately, bisimulation is *not* sound for the ν-calculus. The counterexample is from Pitts and Stark [21]. Consider three functions of type $(\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool}$:

$$\begin{array}{rcl}
t_1 & \stackrel{\mathsf{def}}{=} & \lambda f : \mathtt{name} \to \mathtt{bool} \,.\, \mathtt{true} \\
t_2 & \stackrel{\mathsf{def}}{=} & \nu n \,.\, \nu n' \,.\, \lambda f : \mathtt{name} \to \mathtt{bool} \,.\, f(n) = f(n') \\
t_3 & \stackrel{\mathsf{def}}{=} & \nu n \,.\, \lambda f : \mathtt{name} \to \mathtt{bool} \,.\, \nu n' \,.\, f(n) = f(n')
\end{array}$$

These three terms are all bisimilar, but $t_2 \not\approx_{ctx} t_3$, since the following context distinguishes them:

$$C[\cdot] \stackrel{\mathsf{def}}{=} \mathtt{let}\, F \,=\, \cdot\, \mathtt{in}\, F(\lambda x : \mathtt{name} \,.\, F(\lambda x' : \mathtt{name} \,.\, x = x'))$$

This is the same counterexample that Pitts and Stark use to show that logical relations are incomplete, since logical relations identify *none* of these terms.

Bisimulation is unsound because it is not a congruence. To see why Howe's proof fails, we have to observe that the crucial step in Howe is that $\approx^{\bullet}$ matches let-β reductions. In the ν-calculus we would have to complete the diagram:

$$\begin{array}{ccccc}
\mathtt{let}\, x \,=\, \nu\Delta_1 \,.\, v_1\, \mathtt{in}\, t_1 & \overset{\widehat{\approx^{\bullet}}}{\longleftrightarrow} & \mathtt{let}\, x \,=\, \nu\Delta_2 \,.\, v_2\, \mathtt{in}\, t_2 & \overset{\approx}{\longleftrightarrow} & t_3 \\
\Big\downarrow \tau & & & & \\
\nu\overline{n}_1 \,.\, t_1[v_1/x] & & & &
\end{array}$$

as:

$$\begin{array}{ccccc}
\mathtt{let}\, x \,=\, \nu\Delta_1 \,.\, v_1\, \mathtt{in}\, t_1 & \overset{\widehat{\approx^{\bullet}}}{\longleftrightarrow} & \mathtt{let}\, x \,=\, \nu\Delta_2 \,.\, v_2\, \mathtt{in}\, t_2 & \overset{\approx}{\longleftrightarrow} & t_3 \\
\Big\downarrow \tau & & \Big\downarrow \tau & & \Big\Downarrow \\
\nu\overline{n}_1 \,.\, t_1[v_1/x] & \overset{???}{\longleftrightarrow} & \nu\overline{n}_2 \,.\, t_2[v_2/x] & \overset{\approx}{\longleftrightarrow} & t_3'
\end{array}$$

but in order to complete the diagram we need to know that $\approx^{\bullet}$ is *substitutive on prevalues*:

$$\begin{array}{l}
\text{if } t_1 \approx^{\bullet} t_2 \\
\text{and } \nu\overline{n}_1 \,.\, v_1 \approx^{\bullet} \nu\overline{n}_2 \,.\, v_2 \\
\text{then } \nu\overline{n}_1 \,.\, t_1[v_1/x] \approx^{\bullet} \nu\overline{n}_2 \,.\, t_2[v_2/x]
\end{array}$$

which is not true of the ν-calculus, as witnessed by the counterexample described above.

The best we can manage is to show that bisimulation is sound for the ν-calculus at first order, by showing that bisimulation coincides with Pitts and Stark's logical relation semantics.

Theorem 3.2 (soundness for ν-calculus at first order)
For first order σ, *if* $\Gamma; \Delta \vDash t \approx^{\circ} t' : \sigma$ *then* $\Gamma; \Delta \vDash t \approx_{ctx} t' : \sigma$.

3.7 Comments

We now consider how our bisimulation compares with logical relations. The example given above serves to demonstrate that logical relations are strictly finer than bisimulation, and that they only coincide at first order. The main difference between the two approaches is the view of privacy they adopted. In the bisimulation

approach names are considered private until the secret is leaked by a νn transition, whereas the logical relations use a more overt proof method whereby the environment has access to secrets but can only test using them restrictedly. For terms of first-order type the test values are of ground type and the restrictions imposed by the logical relations are strong enough to disallow any testing with secrets altogether, thus realigning the method with the more covert approach of bisimulation.

There is also an evident analogue of Sangiorgi's context bisimulation [18], which could be formulated for the ν-calculus. In essence, this says that whenever we need to test a λ-abstraction we must consider its behaviour in every context. This could easily be formulated in an lts by allowing suitably typed transitions of the form $v \xrightarrow{f@} f(v)$. This of course, defies our insistence that labels be applicative because the type of f here must be of higher-order than v. In fact, bisimulation on such an lts could easily be shown to be fully abstract but this would, in effect, be no more than a restatement of the **ciu**-theorem of [14] following [9].

Although one obvious reason for bisimulation failing to be a congruence is that the labels in our lts do not provide sufficient distinguishing power, it is difficult to see how the label set could be effectively enlarged without using non-applicative values such as are used for context bisimulation. The problem is terms which contain shared secrets such as $\nu n\,.\,(v_1, v_2)$ where the environment can use v_1 in testing v_2. One possibility would be to add transitions:

$$(v_1, v_2) \xrightarrow{@f@} v_1(f(v_2))$$

together with reductions for the symmetric monoidal structure of pairing. We speculate that such an lts would provide a fully abstract semantics, but we leave this as an open problem.

As an alternative, we extend the ν-calculus to include imperative side-effects, to get the νref-calculus, and show that the extra $@v$ transitions give us full abstraction. This is the subject of the next section.

4 νref-calculus

We have shown that bisimulation is complete (but unsound) for the ν-calculus. In this section we show that adding imperative side-effects to the language allows us to recover a sound and complete semantics.

The reason why assignments allows us to recover completeness is that the counterexamples rely on the fact that n is a 'secret' in terms such as:

$$\lambda f : \mathtt{name} \to \mathtt{bool}\,.\, f(n)$$

despite the fact that some 'foreign' code f is being applied to n. By adding assignment, f can leak the secret name n to the environment.

We believe that any form of side-effect which allows secrets to leak like this will make bisimulation sound and complete, for example call-cc, communication channels or imperative objects. We have chosen to investigate assignment as it is the simplest addition which is still deterministic and terminating.

4.1 Syntax and type rules

Extend the grammar of terms by:

$$t ::= \cdots \mid r := v\,.\,t \mid ?r$$

where r ranges over an infinite set of *references*. These operations allow a name to be written to, or read from, a reference.

We introduce a use-def type system which ensures that a reference is assigned to before it is read. Judgements are now of the form $\Gamma;\Delta;\Theta \vdash t : \sigma$ where Θ is a *reference environment* of the form $r_1, ..., r_n$ (not necessarily distinct). The reference environment lists the names which have been assigned to, and so can be read from.

The new type judgements are:

$$\frac{\Gamma;\Delta;\Theta \vdash v : \mathtt{name} \quad \Gamma;\Delta;\Theta, r \vdash t : \sigma}{\Gamma;\Delta;\Theta \vdash r := v\,.\,t : \sigma} \qquad \frac{}{\Gamma;\Delta;\Theta, r, \Theta' \vdash ?r : \mathtt{name}}$$

For example, we cannot type:

$$\vdash \lambda x : \sigma\,.\,?r$$

since r has not necessarily been initialized, whereas if we add an initialization statement for r then we can type:

$$\vdash (\nu n\,.\,r := n\,.\,\lambda x : \sigma\,.\,?r) : \sigma \to \mathtt{name}$$

4.2 Reduction semantics

Prevalues are now terms of the form:

$$p ::= d\,.\,v \qquad d\,.\,\cdot ::= \nu\overline{n}\,.\,\overline{r} := \overline{n}\,.\,\cdot$$

The reduction semantics for the νref-calculus is defined up to a structural equivalence, following Berry and Boudol's Chemical Abstract Machine [2]. Let $\equiv$ be the least equivalence such that:

$$\begin{array}{rcll}
r := n\,.\,\nu n'\,.\,t & \equiv & \nu n'\,.\,r := n\,.\,t & (n \neq n') \\
r_1 := n_1\,.\,r_2 := n_2\,.\,t & \equiv & r_2 := n_2\,.\,r_1 := n_1\,.\,t & (r_1 \neq r_2) \\
r := n_1\,.\,r := n_2\,.\,t & \equiv & r := n_2\,.\,t & \\
\nu n\,.\,\nu n'\,.\,t & \equiv & \nu n'\,.\,\nu n\,.\,t & \\
r := n\,.\,\mathtt{let}\ v = t\ \mathtt{in}\ t' & \equiv & \mathtt{let}\ v = r := n\,.\,t\ \mathtt{in}\ t' &
\end{array}$$

and:

$$\frac{t_1 \equiv t_2}{\mathcal{E}[t_1] \equiv \mathcal{E}[t_2]}$$

Extend the evaluation contexts to include assignment:

$$\mathcal{E} ::= \cdots \mid r := v\,.\,\mathcal{E}$$

Extend the reduction semantics with a rule for dereferencing:

$$r := n\,.\,?r \quad \xrightarrow{\tau} \quad r := n\,.\,n$$

Since we have extended prevalues, we need to extend the let-β rule:

$$\mathtt{let}\ x = d\,.\,v\ \mathtt{in}\ t \quad \xrightarrow{\tau} \quad d\,.\,t[v/x]$$

Add a structural equivalence rule:

$$\frac{t_1 \equiv t_2 \quad t_2 \xrightarrow{\tau} t_3 \quad t_3 \equiv t_4}{t_1 \xrightarrow{\tau} t_4}$$

The definition of contextual equivalence remains the same, except that the results of a test can include some assignments: $t_1 \approx_{ctx} t_2$ whenever for all closing contexts $\mathcal{C}$ of type $\mathtt{bool}$, we have $\mathcal{C}[t_1] \Rightarrow d_1\,.\,\mathtt{true}$ iff $\mathcal{C}[t_2] \Rightarrow d_2\,.\,\mathtt{true}$.

4.3 Labelled transition system semantics

We need to allow free references in values, so judgements are now of the form $(\Delta;\Theta \vdash p) \xrightarrow{\gamma} (\Delta,\Delta';\Theta \vdash t)$.

Extend the grammar of labels with:

$$\gamma ::= \cdots \mid r{:=}n \mid ?r$$

The new transitions allow a name to be assigned:

$$(\Delta,n,\Delta';\Theta \vdash ()) \xrightarrow{r:=n} (\Delta,n,\Delta';\Theta \vdash r := n\,.\,())$$

and to be read:

$$(\Delta;\Theta,r,\Theta' \vdash ()) \xrightarrow{?r} (\Delta;\Theta,r,\Theta' \vdash ?r)$$

We weaken the side-condition on application to allow the argument to include free references:

$$(\Delta;\Theta \vdash \lambda x:\sigma\,.\,t) \xrightarrow{@v} (\Delta;\Theta \vdash t[v/x]) \qquad (\text{where } \Delta;\Theta \vdash v:\sigma)$$

Transitions are allowed in assignment contexts:

$$\frac{(\Delta;\Theta,r \vdash p) \xrightarrow{\gamma} (\Delta,\Delta';\Theta,r \vdash t)}{(\Delta;\Theta \vdash r := n\,.\,p) \xrightarrow{\gamma} (\Delta,\Delta';\Theta \vdash r := n\,.\,t)}$$

Add a structural equivalence rule:

$$\frac{t_1 \equiv t_2 \quad (\Delta;\Theta \vdash t_2) \xrightarrow{\gamma} (\Delta,\Delta';\Theta \vdash t_3) \quad t_3 \equiv t_4}{(\Delta;\Theta \vdash t_1) \xrightarrow{\gamma} (\Delta,\Delta';\Theta \vdash t_4)}$$

We can define bisimulation between configurations $(\Delta;\Theta \vdash t)$ as we did for the ν-calculus

4.4 Example

We can distinguish the problem cases from the ν-calculus:

$$\begin{aligned} t_1 &\stackrel{\text{def}}{=} \lambda f:\mathtt{name}\to\mathtt{bool}\,.\,\mathtt{true}\\ t_2 &\stackrel{\text{def}}{=} \nu n\,.\,\nu n'\,.\,\lambda f:\mathtt{name}\to\mathtt{bool}\,.\,f(n)=f(n')\\ t_3 &\stackrel{\text{def}}{=} \nu n\,.\,\lambda f:\mathtt{name}\to\mathtt{bool}\,.\,\nu n'\,.\,f(n)=f(n') \end{aligned}$$

It is easy to distinguish t_1 from the others, since we just let f be a function with a side-effect. To distinguish t_2 from t_3 we have:

$$\begin{aligned} t_2 &\xrightarrow{r:=n''} \nu n\,.\,\nu n'\,.\,r := n''\,.\,t_2'\\ &\xrightarrow{\mathtt{copy}} \nu n\,.\,\nu n'\,.\,r := n''\,.\,(t_2',t_2')\\ &\xrightarrow{\mathtt{l}.@\lambda x:\mathtt{name}.r:=x.\mathtt{true}} \nu n\,.\,\nu n'\,.\,r := n'\,.\,(\mathtt{true},t_2')\\ &\xrightarrow{\mathtt{l.discard}} \nu n\,.\,\nu n'\,.\,r := n'\,.\,((),t_2')\\ &\xrightarrow{\mathtt{l}.?r} \nu n\,.\,\nu n'\,.\,r := n'\,.\,(n',t_2')\\ &\xrightarrow{\mathtt{r}.@\lambda x:\mathtt{name}.r:=x.\mathtt{true}} \nu n\,.\,\nu n'\,.\,r := n'\,.\,(n',\mathtt{true})\\ &\xrightarrow{\mathtt{r.discard}} \nu n\,.\,\nu n'\,.\,r := n'\,.\,(n',())\\ &\xrightarrow{\mathtt{r}.?r} \nu n\,.\,\nu n'\,.\,r := n'\,.\,(n',n') \end{aligned}$$

where:

$$t_2' \stackrel{\text{def}}{=} \lambda f:\mathtt{name}\to\mathtt{bool}\,.\,f(n)=f(n')$$

whereas when we try to emulate these transitions in t_3 we end with:

$$\nu n\,.\,\nu n'\,.\,\nu n''\,.\,r := n''\,.\,(n',n'')$$

which are easily distinguished.

4.5 Completeness

Completeness for the νref-calculus follows as it does for the λ-calculus and ν-calculus. The contexts corresponding to the extra transitions for reference manipulation are immediate.

Theorem 4.1 (completeness for νref-calculus)
If $\Gamma;\Delta;\Theta \vDash t \approx_{ctx} t' : \sigma$ *then* $\Gamma;\Delta;\Theta \vDash t \approx^{\circ} t' : \sigma$.

4.6 Soundness

The subject of the next section of this paper is to establish that bisimulation is a congruence for the νref-calculus, from which soundness immediately follows.

Theorem 4.2 (soundness for νref-calculus)
If $\Gamma;\Delta;\Theta \vDash t \approx^{\circ} t' : \sigma$ *then* $\Gamma;\Delta;\Theta \vDash t \approx_{ctx} t' : \sigma$.

4.7 Comments

In the definition of Standard ML [12] a model of references is presented using *configurations* of the form (S,t) where S is a state (a mapping of references to values). In this paper, we have included states in the syntax of terms, so the configuration $(\{\overline{r} \mapsto \overline{v}\},t)$ is modelled by the term $\overline{r} := \overline{v}\,.\,t$. A similar use of syntax to model configuration is used in Ferreira, Hennessy and Jeffrey's [3] lts model of Reppy's [17] configuration-based model of Concurrent ML.

A recent paper of Pitts and Stark [16] also concerns itself with an operational study of locality using references. The difference here is that they use a language of integer references so that all equality tests between local names are definable from primitive operations, such as assignment and equality test on integers. The logical relations presented in [16] are much stronger than those of the ν-calculus, largely because of their non-applicative nature. They are closer in spirit to context bisimulation than to the bisimulations presented here.

We speculate that the techniques here could be adapted to an extension of Pitts and Stark's [16] where references are allowed to all ground types, including references. We leave this as an open problem.

5 Congruence of bisimulation for the νref-calculus

5.1 Active and passive names

To show that bisimulation is a congruence, we need to perform some analysis on the names generated by the prevalues. We define n to be *active* in $(\Delta;\Theta \vdash t)$ if there is some sequence of transitions $\overline{\gamma}$ not containing n such that $(\Delta;\Theta \vdash t) \stackrel{\overline{\gamma}}{\Longrightarrow} \stackrel{\mathtt{l}.n}{\Longrightarrow} (\Delta,\Delta';\Theta \vdash t')$, and *passive* otherwise.

Intuitively, if a name is passive in a term then it is a secret which the term never reveals. Unfortunately, this intuition does not hold for the ν-calculus, where we can construct terms:

$$\begin{aligned} F &\stackrel{\text{def}}{=} \lambda f:\mathtt{name}\to\mathtt{bool}\,.\,\nu n'\,.\,\mathtt{if}\ f(n')=f(n)\ \mathtt{then}\ n'\ \mathtt{else}\ n\\ f &\stackrel{\text{def}}{=} \lambda x:\mathtt{name}\,.\,x=n \end{aligned}$$

then we have n is passive in F and f, but is active in $F(f)$. This is not only very counterintuitive, but strikes at the heart of the problem for finding a fully abstract model for ν-calculus. The logical relations approach fails to be complete because when new names are generated it must be guessed whether the names are active (global) or passive (secret). The environment is then allowed to test with secret names provided they occur passively in the test value. During testing though the names may change their status from passive to active so no consistent guess can be made.

In comparison, in the νref-calculus, although n is passive in f, it is *not* passive in F since we have the reduction:

$$\begin{array}{rll} F & \xrightarrow{@\lambda x{:}\mathtt{name}.r := x.\mathtt{true}} & \nu n' \,.\, r := n \,.\, n' \\ & \xrightarrow{\mathtt{discard}} & \nu n' \,.\, r := n \,.\, () \\ & \xrightarrow{?r} & \nu n' \,.\, r := n \,.\, n \\ & \xrightarrow{n} & \nu n' \,.\, r := n \,.\, () \end{array}$$

We can generalize this to get the following proposition, which is *not* true of the ν-calculus:

Proposition 5.1 *If n is passive in t and v then n is passive in $t[v/x]$.*

Proof: A very long and involved proof, which we will appear in the full version of this paper. □

5.2 Overt bisimulation

Motivated by the distinction between active and passive names, we present an alternative bisimulation for the νref-calculus, which is more complex, but turns out to be more suitable for Howe-style proof.

This relation is inspired by Pitts and Stark's logical relations semantics although there are some subtle differences, which we will discuss later.

A type-indexed family of relations R^Π (where Π is a name environment) is an *overt simulation* if:

1. We can complete the following diagram:

$$\begin{array}{ccc} (\Delta \vdash t_1) & \overset{R^\Pi}{\longleftrightarrow} & (\Delta \vdash t_2) \\ \alpha \big\downarrow & & \\ (\Delta, \Delta' \vdash t_1') & & \end{array}$$

   when Δ' is disjoint from Π as:

$$\begin{array}{ccc} (\Delta \vdash t_1) & \overset{R^\Pi}{\longleftrightarrow} & (\Delta \vdash t_2) \\ \alpha \big\downarrow & & \hat{\alpha} \big\Downarrow \\ (\Delta, \Delta' \vdash t_1') & \overset{R^\Pi}{\longleftrightarrow} & (\Delta, \Delta' \vdash t_2') \end{array}$$

2. If $(\Delta \vdash \nu n \,.\, p_1)\ R^\Pi\ (\Delta \vdash p_2)$ then either:
   (a) $p_2 \equiv \nu n \,.\, p_3$ and $(\Delta, n \vdash p_1)\ R^\Pi\ (\Delta, n \vdash p_3)$, or
   (b) $(\Delta \vdash p_1)\ R^{\Pi, n}\ (\Delta \vdash p_2)$.

3. If $(\Delta \vdash p_1)\ R^\Pi\ (\Delta \vdash p_2)$ and $(\Delta \vdash p_1) \xrightarrow{\iota.n} (\Delta \vdash p_1')$ then $n \in \Delta$.

Let $\approx_o^\Pi$ be the largest overt bisimulation. We shall write $\approx_o$ when Π is empty.

Intuitively, the definition of an overt bisimulation says:

1. $\approx_o^\Pi$ is a bisimulation,
2. If $\Delta \vdash \nu n \,.\, p_1 \approx_o^\Pi p_2$ then either:
   (a) n is active in p_1, so p_2 has to match it by having an appropriate name binder (which is added to the active name environment Δ), or
   (b) n is passive in p_1, so p_2 can match it by ignoring the name (which is added to the passive name environment Π).
3. If $\Delta \vdash p_1 \approx_o^\Pi p_2$ then Δ contains all the active names of p_1.

Since an overt simulation is a simulation, it is easy to see that $\approx_o$ is a finer relation than $\approx$. In fact, we can show that overt bisimulation coincides with bisimulation.

Proposition 5.2 *$\approx$ is the same as $\approx_o$*

Proof: Define $\Delta \vDash t_1 \approx^\Pi t_2$ whenever $\Delta \vDash \nu\Pi \,.\, t_1 \approx \nu\Pi \,.\, t_2$ and all the names in Π are passive in t_1 and t_2. It is routine to verify that this is an overt bisimulation, and that it coincides with bisimulation when Π is empty. □

5.3 Congruence of overt bisimulation

The proof that overt bisimulation is a congruence uses Howe's technique, but the definition of $\approx^\bullet$ is rather more complex, since we have to allow names to move between the passive and active name environments.

Define $\approx_o^{\Pi^\bullet}$ by two rules (for any name environment Ξ):

$$\frac{\Gamma;\Delta,\Xi;\Theta \vDash t_1 \widehat{\approx_o^{\Pi^\bullet}} t_2 \qquad \Gamma;\Delta;\Theta \vDash t_2 \approx_o^{\Pi,\Xi^\circ} t_3}{\Gamma;\Delta;\Theta \vDash t_1 \approx_o^{\Pi,\Xi^\bullet} t_3}$$

and:

$$\frac{\Gamma;\Delta;\Theta \vDash \nu n \,.\, t_1 \widehat{\approx_o^{\Pi,n^\bullet}} t_2 \qquad \Gamma;\Delta;\Theta \vDash t_2 \approx_o^{\Pi^\circ} t_3}{\Gamma;\Delta;\Theta \vDash \nu n \,.\, t_1 \approx_o^{\Pi^\bullet} t_3}$$

Then we can verify that $\approx_o^{\Pi^\bullet}$ is substitutive on prevalues, up to strong bisimulation:

Proposition 5.3

$$\begin{array}{l} \textit{if } t_1 \approx_o^{\Pi^\bullet} t_2 \\ \textit{and } d \,.\, v_1 \approx_o^{\Pi^\bullet} d \,.\, v_2 \\ \textit{then } d \,.\, t_1[v_1/x] \sim\!\approx_o^{\Pi^\bullet} d \,.\, t_2[v_1/x] \end{array}$$

Proof: By structural induction on the prevalues. This proof relies on substitution preserving passivity (Propn 5.1). □

We can then show that $\approx^\bullet$ is a bisimulation up to $(\sim, =)$ [19], from which it is routine to show that overt bisimulation, and hence bisimulation, is a congruence.

Theorem 5.4 *$\approx^\circ$ is a congruence for the νref-calculus.*

5.4 Comments

Earlier in the paper we described the logical relations of [15] as an *overt* proof technique for ν-calculus. We can see now that there are similarities between our overt bisimulation and the logical relations. In particular, both techniques make use of a predicate to track the private names of terms under test. In the logical relations the predicate takes the form of a partial injection between the free names of terms—secret names are those not in the domain (or range) of this injection.

The key point is that when a new name is to be generated, in both overt bisimulation and logical relations, it must be guessed whether the name will be active or passive. Where the two approaches differ greatly however is in the tests allowed in the *@v* transitions. Logical relations allow the environment to use secrets passively, even though, morally, they have no knowledge of them. Overt bisimulations forbid this and insist that a secret is a secret and until the environment knows the secret it cannot test with it at all. It would be interesting to show that these two approaches coincide for the νref-calculus. In light of Proposition 5.1 there is strong evidence to suggest that they do, but we leave this as an open problem.

References

[1] K. L. Bernstein and E. W. Stark. Operational semantics of a focussing debugger. In *Proc. MFPS 95*, number 1 in Electronic Notes in Comp. Sci. Springer-Verlag, 1995.

[2] G. Berry and G. Boudol. The chemical abstract machine. In *Proc. 17th Ann. Symp. Principles of Programming Languages*, 1990.

[3] W. Ferreira, M. Hennessy, and A.S.A. Jeffrey. A theory of weak bisimulation for core CML. In *Proc. ACM SIGPLAN Int. Conf. Functional Programming*, pages 201–212. ACM Press, 1996. To appear in *J. Functional Programming*.

[4] A. Gordon and M. Abadi. A calculus for cryptographic protocols: The spi calculus. Research Report 149, Digital Equipment Corporation Systems Research Center, 1998. To appear in *Information and Computation*.

[5] A. Gordon and L. Cardelli. Mobile ambients. In *Proc. FoSSaCS '98*, LNCS. Springer-Verlag, 1998.

[6] A. D. Gordon. Bisimilarity as a theory of functional programming. In *Proc. MFPS 95*, number 1 in Electronic Notes in Comp. Sci. Springer-Verlag, 1995.

[7] A. D. Gordon. Nominal calculi for security and mobility. In *Proc. DARPA Workshop on Foundations for Secure Mobile Code*, pages 10–14, 1997.

[8] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[9] F. Honsell, I.A. Mason, S. Smith, and C. Talcott. A variable typed logic of effects. *Inform. and Comput.*, 119(1):55–90, 1995.

[10] D. Howe. Equality in lazy computation systems. In *Proc. LICS '89*, pages 198–203. IEEE Computer Society Press, 1989.

[11] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proceses. *Inform. and Comput.*, 100(1):1–77, 1992.

[12] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[13] E. Moggi. Notions of computation and monads. *Inform. and Comput.*, 93:55–92, 1991.

[14] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Proc. MFCS 93*, pages 122–141. Springer-Verlag, 1993. LNCS 711.

[15] A. M. Pitts and I. D. B. Stark. On the observable properties of higher order functions that dynamically create local names (preliminary report). In *Workshop on State in Programming Languages, Copenhagen, 1993*, pages 31–45. ACM SIGPLAN, 1993. Yale Univ. Dept. Computer Science Technical Report YALEU/DCS/RR-968.

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

[17] J. Reppy. *Higher-Order Concurrency*. Ph.D thesis, Cornell Univ., 1992.

[18] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms*. Ph.D thesis, LFCS, Edinburgh Univ., 1992.

[19] D. Sangiorgi and R. Milner. Techniques of 'weak bisimulation up to'. In *Proc. CONCUR 92*. Springer Verlag, 1992. LNCS 630.

[20] P. Sewell. From rewrite rules to bisimulation congruences. In *Proc. CONCUR '98*, volume 1466 of *LNCS*, pages 269–284. Springer-Verlag, 1998.

[21] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994. Also published as Technical Report 363, University of Cambridge Computer Laboratory.

[22] A. Wright and M. Felleisen. A syntactic approach to type soundness. Technical report TR91-160, Dept. of Comp. Sci., Rice Univ., 1991.