Notes on ambient rho

A reflective higher-order calculus fully-abstractly encodes ambients

L.G. MEREDITH AND MATTHIAS RADESTOCK

Abstract. Comm rule with splitting allows to encode out rule of ambients. Lift and drop allows to encode in rule of ambients. Communication allows for open.

1. Introduction

Blah, blah, blah...

1.1. **Overview and contributions.** We illustrate a calculation of rates.

2. The calculus

2.0.1. *Notation.* We let P, Q, R range over processes and x, y, z range over names.

ρ-calculus	P, Q	$::=$	0	null process
		$\mid$	$x(y)\,.\,P$	input
		$\mid$	$x\langle\!\langle P \rangle\!\rangle$	lift
		$\mid$	$\urcorner x \ulcorner$	drop
		$\mid$	$P \mid Q$	parallel
	x, y	$::=$	$\ulcorner P \urcorner$	quote

2.0.2. *Input and output.* The input constructor is standard for an asynchronous name-passing calculus. Input blocks its continuation from execution until it receives a communication. Lift is a form of output which – because the calculus is asynchronous – is allowed no continuation. It also affords a convenient syntactic sugar, which we define here.

$$x[y] \triangleq x\langle\!\langle \urcorner y \ulcorner \rangle\!\rangle$$

Key words and phrases. concurrency, message-passing, process calculus, reflection.

2.1. **Free and bound names.** The syntax has been chosen so that a binding occurrence of a name is sandwiched between round braces, $(\cdot)$. Thus, the calculation of the free names of a process, P, denoted $\mathcal{FN}(P)$ is given recursively by

$$\begin{aligned}
\mathcal{FN}(0) &= \emptyset \\
\mathcal{FN}(x(y)\,.\,P) &= \{x\} \cup (\mathcal{FN}(P) \setminus \{y\}) \\
\mathcal{FN}(x\langle\!|P|\!\rangle) &= \{x\} \cup \mathcal{FN}(P) \\
\mathcal{FN}(P \mid Q) &= \mathcal{FN}(P) \cup \mathcal{FN}(Q) \\
\mathcal{FN}(\ulcorner x \urcorner) &= \{x\}
\end{aligned}$$

An occurrence of x in a process P is *bound* if it is not free. The set of names occurring in a process (bound or free) is denoted by $\mathcal{N}(P)$.

2.2. **Structural congruence.** The *structural congruence* of processes, noted $\equiv$, is the least congruence, containing α-equivalence, $\equiv_\alpha$, that satisfies the following laws:

$$\begin{aligned}
P \mid 0 &\equiv P \equiv 0 \mid P \\
P \mid Q &\equiv Q \mid P \\
(P \mid Q) \mid R &\equiv P \mid (Q \mid R)
\end{aligned}$$

2.3. **Name equivalence.** We now come to one of the first real subtleties of this calculus. Both the calculation of the free names of a process and the determination of structural congruence between processes critically depend on being able to establish whether two names are equal. In the case of the calculation of the free names of an input-guarded process, for example, to remove the bound name we must determine whether it is in the set of free names of the continuation. Likewise, structural congruence includes α-equivalence. But, establishing α-equivalence between the processes $x(z)\,.\,w\langle\!|y[z]|\!\rangle$ and $x(v)\,.\,w\langle\!|y[v]|\!\rangle$, for instance, requires calculating a substitution, e.g. $x(v)\,.\,w\langle\!|y[v]|\!\rangle\{z/v\}$. But this calculation requires, in turn, being able to determine whether two names, in this case the name in the object position of the output, and the name being substituted for, are equal.

As will be seen, the equality on names involves structural equivalence on processes, which in turn involves alpha equivalence, which involves name equivalence. This is a subtle mutual recursion, but one that turns out to be well-founded. Before presenting the technical details, the reader may note that the grammar above enforces a strict alternation between quotes and process constructors. Each question about a process that involves a question about names may in turn involve a question about processes, but the names in the processes the next level down, as it were, are under fewer quotes. To put it another way, each 'recursive call' to name equivalence will involve one less level of quoting, ultimately bottoming out in the quoted zero process.

Let us assume that we have an account of (syntactic) substitution and α-equivalence upon which we can rely to formulate a notion of name equivalence, and then bootstrap our notions of substitution and α-equivalence from that. We take name equivalence, written $\equiv_N$, to be the smallest equivalence relation generated by the following rules.

$$\frac{}{\ulcorner\urcorner x \ulcorner\urcorner \equiv_N x} \qquad \text{(Quote-drop)}$$

$$\frac{P \equiv Q}{\ulcorner P \urcorner \equiv_N \ulcorner Q \urcorner} \qquad \text{(Struct-equiv)}$$

2.4. **Syntactic substitution.** Now we build the substitution used by α-equivalence. We use $Proc$ for the set of processes, $\ulcorner Proc \urcorner$ for the set of names, and $\{\vec{y}/\vec{x}\}$ to denote partial maps, $s : \ulcorner Proc \urcorner \to \ulcorner Proc \urcorner$. A map, s lifts, uniquely, to a map on process terms, $\widehat{s} : Proc \to Proc$ by the following equations.

$$\begin{aligned}
(0)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} &= 0 \\
(R \mid S)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} &= (R)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} \mid (S)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} \\
(x(y) \,.\, R)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} &= (x)\{\ulcorner Q \urcorner/\ulcorner P \urcorner\}(z) \,.\, ((R\{z/y\})\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\}) \\
(x\langle\!|R|\!\rangle)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} &= (x)\{\ulcorner Q \urcorner/\ulcorner P \urcorner\}\langle\!|R\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\}|\!\rangle \\
(\urcorner x \ulcorner)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} &= \begin{cases} \urcorner\ulcorner Q \urcorner\ulcorner & x \equiv_N \ulcorner P \urcorner \\ \urcorner x \ulcorner & otherwise \end{cases}
\end{aligned}$$

where

$$(x)\{\ulcorner Q \urcorner/\ulcorner P \urcorner\} = \begin{cases} \ulcorner Q \urcorner & x \equiv_N \ulcorner P \urcorner \\ x & otherwise \end{cases}$$

and z is chosen distinct from $\ulcorner P \urcorner$, $\ulcorner Q \urcorner$, the free names in Q, and all the names in R. Our α-equivalence will be built in the standard way from this substitution.

But, given these mutual recursions, the question is whether the calculation of $\equiv_N$ (respectively, $\equiv$, $\equiv_\alpha$) terminates. To answer this question it suffices to formalize our intuitions regarding level of quotes, or quote depth, $\#(x)$, of a name x as follows.

$$\begin{aligned}
\#(\ulcorner P \urcorner) &= 1 + \#(P) \\
\#(P) &= \begin{cases} max\{\#(x) : x \in \mathcal{N}(P)\} & \mathcal{N}(P) \neq \emptyset \\ 0 & otherwise \end{cases}
\end{aligned}$$

The grammar ensures that $\#(\ulcorner P \urcorner)$ is bounded. Then the termination of $\equiv_N$ (respectively, $\equiv$, $\equiv_\alpha$) is an easy induction on quote depth.

2.5. **Semantic substitution.** The substitution used in α-equivalence is really only a device to formally recognize that binding occurrences do not depend on the specific names. It is not the engine of computation. The proposal here is that while synchronization is the driver of that engine, the real engine of computation is a semantic notion of substitution that recognizes that a dropped name is a request to run a process. Which process? Why the one whose code has been bound to the name being dropped. Formally, this amounts to a notion of substitution that differs from syntactic substitution in its application to a dropped name.

$$(\urcorner x \ulcorner)\{\widehat{\ulcorner Q \urcorner/\ulcorner P \urcorner}\} = \begin{cases} Q & x \equiv_N \ulcorner P \urcorner \\ \urcorner x \ulcorner & otherwise \end{cases}$$

In the remainder of the paper we will refer to semantic and syntactic substitutions simply as substitutions and rely on context to distinguish which is meant. Similarly, we will abuse notation and write $\{y/x\}$ for $\widehat{\{y/x\}}$.

Finally equipped with these standard features we can present the dynamics of the calculus.

2.6. **Operational Semantics.** The reduction rules for ρ-calculus are

$$\frac{x_0 \equiv_N x_1}{x_0(\!|Q \mid R|\!) \mid x_1(y) \,.\, P \to x_0(\!|R|\!) \mid P\{{\ulcorner Q \urcorner}/{y}\}} \tag{Comm}$$

In addition, we have the following context rules:

$$\frac{P \to P'}{P \mid Q \to P' \mid Q} \tag{Par}$$

$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q} \tag{Equiv}$$

The context rules are entirely standard and we do not say much about them, here. The communication rule does what was promised, namely make it possible for agents to synchronize and communicate processes packaged as names. Additionally, it allows us to encode ambient behavior.

$$\begin{array}{rcl} x(\!|P|\!) \mid x(y) \,.\, z(\!|Q \mid x(\!|\ulcorner y \urcorner|\!)|\!) & \to^* & z(\!|Q \mid x(\!|P|\!)|\!) \\ x(\!|y(\!|x(\!|Q|\!)|\!) \mid R|\!)|\!) \mid x(z).\ulcorner z \urcorner \mid y(w).\ulcorner w \urcorner \mid x(u) \,.\, y(\!|\ulcorner u \urcorner|\!) & \to^* & x(\!|R|\!) \mid y(\!|Q|\!) \\ x(\!|P|\!) \mid x(y).\ulcorner y \urcorner \mid Q & \to & P \mid Q \end{array}$$

3. Replication

As mentioned before, it is known that replication (and hence recursion) can be implemented in a higher-order process algebra [**?**]. As our first example of calculation with the machinery thus far presented we give the construction explicitly in the ρ-calculus.

$$\begin{array}{rcl} D(x) & \triangleq & x(y) \,.\, (x[y] \mid \ulcorner y \urcorner) \\ !P(x) & \triangleq & x(\!|D(x) \mid P|\!) \mid D(x) \end{array}$$

Of course, this encoding, as an implementation, runs away, unfolding $!P$ eagerly. A lazier and more implementable replication operator, restricted to input-guarded processes, may be obtained as follows.

$$!u(v) \,.\, P \triangleq x(\!|u(v) \,.\, (D(x) \mid P)|\!) \mid D(x)$$

4. Bisimulation

Having taken the notion of restriction out of the language, we carefully place it back into the notion of observation, and hence into the notion of program equality, i.e. bisimulation. That is, we parameterize the notion of barbed bisimulation by a set of names over which we are allowed to set the barbs. The motivation for this choice is really comparison with other calculi. The set of names of the ρ-calculus is *global*. It is impossible, in the grammar of processes, to guard terms from being placed into contexts that can potentially observe communication. So, we provide a place for reasoning about such limitations on the scope of observation in the theory of bisimulation.

Definition 4.0.1. An *observation relation*, $\downarrow_{\mathcal{N}}$, over a set of names, $\mathcal{N}$, is the smallest relation satisfying the rules below.

$$\frac{y \in \mathcal{N},\ x \equiv_N y}{x[v] \downarrow_{\mathcal{N}} x} \qquad \text{(Out-barb)}$$

$$\frac{P \downarrow_{\mathcal{N}} x \text{ or } Q \downarrow_{\mathcal{N}} x}{P \mid Q \downarrow_{\mathcal{N}} x} \qquad \text{(Par-barb)}$$

We write $P \Downarrow_{\mathcal{N}} x$ if there is Q such that $P \Rightarrow Q$ and $Q \downarrow_{\mathcal{N}} x$.

Notice that $x(y)\ .\ P$ has no barb. Indeed, in ρ-calculus as well as other asynchronous calculi, an observer has no direct means to detect if a message sent has been received or not.

Definition 4.0.2. An *$\mathcal{N}$-barbed bisimulation* over a set of names, $\mathcal{N}$, is a symmetric binary relation $\mathcal{S}_{\mathcal{N}}$ between agents such that $P\ \mathcal{S}_{\mathcal{N}} Q$ implies:

(1) If $P \rightarrow P'$ then $Q \Rightarrow Q'$ and $P'\ \mathcal{S}_{\mathcal{N}} Q'$.
(2) If $P \downarrow_{\mathcal{N}} x$, then $Q \Downarrow_{\mathcal{N}} x$.

P is $\mathcal{N}$-barbed bisimilar to Q, written $P \dot{\approx}_{\mathcal{N}} Q$, if $P\ \mathcal{S}_{\mathcal{N}} Q$ for some $\mathcal{N}$-barbed bisimulation $\mathcal{S}_{\mathcal{N}}$.

5. Conclusions and future work

Blah, blah, blah, ambients, blah...

Acknowledgments. The authors wish to acknowledge sleeplessness and chocolate.