A General Name Binding Mechanism*

Michele Boreale[1], Maria Grazia Buscemi[2], and Ugo Montanari[2]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy
boreale@dsi.unifi.it
[2] Dipartimento di Informatica, Università di Pisa, Italy
{buscemi, ugo}@di.unipi.it

Abstract. We study fusion and binding mechanisms in name passing process calculi. To this purpose, we introduce the U-Calculus, a process calculus with no I/O polarities and a unique form of binding. The latter can be used both to control the scope of fusions and to handle new name generation. This is achieved by means of a simple form of typing: each bound name x is annotated with a set of *exceptions*, that is names that cannot be fused to x. The new calculus is proven to be more expressive than pi-calculus and Fusion calculus separately. In U-Calculus, the syntactic nesting of name binders has a semantic meaning, which cannot be overcome by the ordering of name extrusions at runtime. Thanks to this mixture of static and dynamic ordering of names, U-Calculus admits a form of labelled bisimulation which is a congruence. This property yields a substantial improvement with respect to previous proposals by the same authors aimed at unifying the above two languages. The additional expressiveness of U-Calculus is also explored by providing a uniform encoding of mixed guarded choice into the choice-free sub-calculus.

1 Introduction

Name binding is a key issue in many languages for the design of distributed and mobile systems based on message-passing. This is certainly the case for foundational calculi like pi-calculus [5,6] and Fusion [10], but the relevance of name binding extends also to languages like Biztalk [4] and Highwire [3], oriented towards web services. Fusion extends the pi-calculus by introducing *fusions*, i.e. name equivalences that, when applied onto a term, have the effect of a (possibly non-injective) name substitution. Fusions conveniently formalise, e.g., forwarders for objects that migrate among locations [2], or forms of pattern matching between pairs of messages [3].

While Fusion is presented in [10] as a generalisation of the pi-calculus, the authors prove in the paper [1] that no satisfactory semantic embedding exists of pi-calculus into Fusion. In particular, Fusion ignores the issue of name unicity. In pi-calculus, names declared through the restriction operator are unique, in the sense that they cannot be identified with any other name. In Fusion, the binder

* Research partially supported by IST FET Global projects *PROFUNDIS* IST-2001-33100 and *MIKADO* IST-2001-32222.

R. De Nicola and D. Sangiorgi (Eds.): TGC 2005, LNCS 3705, pp. 61–74, 2005.

(x) can be used to control the scope of fusions, but not to forbid them: names are like logical variables, i.e., unification always succeeds. In [1], we introduce D-Fusion, a calculus with two binders, ν and λ, which extend the binders of pi-calculus and Fusion. We show that D-Fusion is strictly more expressive than both pi-calculus and Fusion. In particular, we prove that both Fusion and pi-calculus can be uniformly mapped into D-Fusion, and exhibit an encoding of mixed guarded choice into the choice-free fragment of D-Fusion.

In D-Fusion, however, constraints on name fusions are totally determined by the extrusion ordering of names at runtime: the fact that a λ-name x will be fusable to a ν-name y depends on whether x will be extruded after y or before y. In other words, fusions cannot be constrained statically. As we explain below, this causes bisimilarity defined on the labelled transition system not to be a congruence. As a consequence, in D-Fusion one is forced to work with barbed congruence, which lacks adequate proof techniques.

In this paper we introduce the U-Calculus, a process calculus with no I/O polarities and a unique form of binding. In U-Calculus, the syntactic nesting of name binders has a semantic meaning, which cannot be overcome by the ordering of name extrusions at runtime. Thanks to this mixture of static and dynamic ordering of names, U-Calculus labelled bisimulation is a congruence.

To understand why a static ordering of names is useful, we can reason as follows. Assume that an agent has a free name x and a ν-bound name y. Names x and y cannot be fused in any reasonable semantics. For example, in open pi-calculus [13], one has $(\nu y)\,[x = y]P \sim \mathbf{0}$. This can be expressed by the following expansion law, which holds true because communication between the two prefixes is forbidden (here we use polarities for the sake of readability; $a\langle x\rangle$ is Fusion's free input, $\overline{a}\langle y\rangle$ is output, and $\sim$ is labelled bisimilarity):

$$P \stackrel{\triangle}{=} (\nu y)(a\langle x\rangle|\overline{a}\langle y\rangle) \;\sim\; (\nu y)(a\langle x\rangle.\overline{a}\langle y\rangle + \overline{a}\langle y\rangle.a\langle x\rangle) \stackrel{\triangle}{=} Q.$$

Now, suppose P and Q above are plugged into a context $(\lambda x)[\cdot]$. If $(\lambda x)P$ ν-extrudes y before λ-extruding x, fusion of x and y will be allowed. This must be the case, at least, if one keeps the traditional scope-extrusion law, which is common to Pi, Fusion and D-Fusion. In fact, scope extrusion allows the binders (λx) and (νy) in $(\lambda x)P$ to be freely swapped. This swapping makes the syntactic ordering of binders immaterial. So $(\lambda x)P$ is equivalent to $a(x)|(\nu y)\overline{a}\langle y\rangle$, where the bound input $a(x)$ is just the same as $(\lambda x)a\langle x\rangle$. In other words:

$$(\lambda x)P \;\sim\; (\lambda x)Q + \tau \;\not\sim\; (\lambda x)Q.$$

Thus, in D-Fusion, plugging an agent into a λ-context may trigger additional communication capabilities, making two agents in the λ-context not bisimilar, when the two original agents were so. Note that this is true even if we require that $\sim$ be closed under all substitutions, in sharp contrast with both open pi-calculus [13] and Fusion. In these calculi, the problem does not arise simply because free input and restriction do not coexist.

A static ordering of name binders solves the problem. In U-Calculus, the syntactic nesting $(\lambda x)(\nu y)$ forbids the fusion of the two names in any case. Op-

erationally, when the extrusion of y takes place under (λx), name x is decorated with an *exception* y, yielding $(\lambda x : y)$. This indicates that the fusion between x and y will never be allowed, and so it holds that $(\lambda x)P \sim (\lambda x)Q$. Semantically, this fact has consequences on the scope extrusion laws. In particular, we have the following new swapping law:

$$(\lambda x)(\nu y)R \sim (\nu y)(\lambda x : y)R.$$

Incidentally, a simple generalisation of the exception types allows to operationally unify the mechanism of λ- and ν-binding. In fact, a ν-binder is just a λ-binder where *all* names free at the moment of extrusion are considered as exceptions. This is indicated by a new type ω, as in $(\lambda x : \omega)$. With this notation, ν and λ enjoy a uniform treatment. As a consequence, the U-Calculus achieves minimal syntax and operational rules.

The expressive power of the U-Calculus is essentially the same as D-Fusion's: also for the U-Calculus we can provide uniform mappings of both pi-calculus and Fusion, and a uniform encoding of mixed guarded choice into the choice-free sub-calculus. In our assesment of the expressive power we shall rely on barbed bisimilarity [12], when this is technically convenient.

The rest of the paper is organised as follows. In Section 2 we introduce the U-Calculus, its operational semantics and a notion of open bisimulation. In Section 3 we show that U-Calculus is strictly more expressive than both pi-calculus and Fusion. We further explore this expressiveness gap in Section 4, by encoding mixed guarded choice into the choice-free calculus. Section 5 contains a few concluding remarks.

2 The U-Calculus

Syntax. We consider a countable set of names $\mathcal{N}$ ranged over by $a, b, \ldots, u, v, ..., z$. We write $\widetilde{x}$ for a finite tuple $(x_1, \ldots, x_n)$ of names. The set $\mathcal{U}$ of U-Calculus *processes*, ranged over by $P, Q, \ldots$, is defined by the syntax:

$$P ::= \mathbf{0} \mid a\,\widetilde{v}.P \mid P|P \mid P + P \mid [x = y]P \mid !P \mid (\lambda x : T)\,P.$$

Types T are defined as:

$$T ::= N \mid \omega,$$

where $N \subseteq_{fin} \mathcal{N}$ and ω is a constant. The intended meaning of $(\lambda x : T)$ is that x cannot be fused with any name in T. In particular, ω stands for 'any name' free at the moment of extrusion, thus $(\lambda x : \omega)\,P$ corresponds to declaring x fresh. We will often abbreviate $(\lambda x : \omega)$ as (νx) and $(\lambda x : \emptyset)$ as (λx). By $(\lambda \widetilde{x} : \widetilde{T})$ we will denote $(\lambda x_1 : T_1) \cdots (\lambda x_n : T_n)$, where it is assumed that $x_i \in T_j$ implies $i < j$, for $i, j = 1, \cdots, n$. We will also adopt the convention that $(\lambda x : T)\,P \,|\, Q$ stands for $((\lambda x : T)\,P) \,|\, Q$.

The occurrences of x in $(\lambda x : T)\,P$ are *bound*, thus notions of *free names* and *bound names* of a process P arise as expected and are denoted by $\mathrm{fn}(P)$ and

bn(P), respectively. The notion of *alpha-equivalence* also arises as expected. In the rest of the paper we will identify alpha-equivalent processes. A *context* $C[\cdot]$ is a process with a hole that can be filled with any process P, thus yielding a process $C[P]$.

Note that we consider one kind of prefix, thus ignoring polarities. However, a sub-calculus with polarities can be easily retrieved, as we will show later in this section.

Notation

- $T + T' \stackrel{\text{def}}{=} N \cup N'$ if $T = N$ and $T' = N'$, $T + T' \stackrel{\text{def}}{=} \omega$ if $T = \omega$ or $T' = \omega$. We abbreviate $T + \{y\}$ as $T + y$.
- $T - y \stackrel{\text{def}}{=} N \setminus \{y\}$ if $T = N$, $T - y \stackrel{\text{def}}{=} \omega$ if $T = \omega$.
- $T \sqcap N \stackrel{\text{def}}{=} N' \cap N$ if $T = N'$, $T \sqcap N \stackrel{\text{def}}{=} N$ if $T = \omega$.
- Predicate $y \,\mathcal{E}\, T$ is defined as follows:

$$y \,\mathcal{E}\, T \;\Leftrightarrow\; T = \omega \text{ or } (T = N \text{ and } y \in N)$$

The above notations are extended to tuples $\widetilde{T}$ as expected. For instance, $\widetilde{T} \sqcap N \stackrel{\text{def}}{=} T_1 \sqcap N, \cdots, T_n \sqcap N$, if $\widetilde{T} = T_1, \cdots, T_n$. For $\widetilde{x} = (x_1, \cdots, x_n)$, $\widetilde{T} = (T_1, \cdots, T_n)$, and $\widetilde{N} = (N_1, \cdots, N_n)$, by $\widetilde{x} : \widetilde{T} \sqcap \widetilde{N}$ we denote $x_1 : T_1 \sqcap N_1, \cdots, x_n : T_n \sqcap N_n$.

Operational Semantics. For R a binary relation over $\mathcal{N}$, let $R^\star$ denote the reflexive, symmetric and transitive closure of R with respect to $\mathcal{N}$. We use σ, σ' to range over substitutions, i.e. finite partial functions from $\mathcal{N}$ onto $\mathcal{N}$. The domain of σ is denoted by $\mathrm{dom}(\sigma)$. We denote by $t\sigma$ the result of applying σ onto a term t. Given a tuple of names $\widetilde{x}$, we define $\sigma_{|\widetilde{x}}$ as $\sigma \cap (\widetilde{x} \times \mathcal{N})$.

Below, we define fusions, that is, name equivalences. These arise as the result of equating two tuples of names in a synchronisation.

Definition 1 (fusions). *We let $\phi, \chi, \ldots$ range over* fusions*, that is total equivalence relations on $\mathcal{N}$ with only finitely many non-singleton equivalence classes. We let:*

- $\mathrm{n}(\phi)$ *denote* $\{x : x \,\phi\, y$ *for some* $y \neq x\}$*;*
- τ *denote the identity fusion (thus,* $\mathrm{n}(\tau) = \emptyset$*);*
- ϕ_{-z} *denote* $(\phi - (\{z\} \times \mathcal{N} \cup \mathcal{N} \times \{z\}))^\star$*;*
- $\{\widetilde{x} = \widetilde{y}\}$ *denote* $\{(x_1, y_1), \ldots, (x_n, y_n)\}^\star$*, where* $\widetilde{x} = x_1, \ldots x_n$ *and* $\widetilde{y} = y_1, \ldots y_n$*;*
- $\phi[x]$ *denote the equivalence class of x in ϕ.*

Definition 2. *Let σ be a substitution. Then, σ is a* substitutive effect *of a fusion ϕ iff* $\forall\, x, y : x \,\phi\, y \;\Leftrightarrow\; x\,\sigma = y\,\sigma$ *and* $\forall\, x, y : \sigma(x) = y \Rightarrow x \,\phi\, y$.

We introduce below a concept of *distinction*, akin to [13]. The purpose of distinctions is to keep track of those name fusions that have to be forbidden.

Definition 3 (distinctions). *A* distinction D *is a tuple* $x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n$, *written* $\widetilde{x} : \widetilde{T}$, *with* ω *not in* $\widetilde{T}$, *up to permutations and up to the law:*

$$\widetilde{x} : \widetilde{T}, w : T_1, w : T_2 = \widetilde{x} : \widetilde{T}, w : T_1 + T_2.$$

Let $D = \widetilde{x} : \widetilde{T}$ *and* $D' = \widetilde{x}' : \widetilde{T}'$ *be two distinctions. Then:*

- $D, D' \stackrel{\text{def}}{=} \widetilde{x}\widetilde{x}' : \widetilde{T}\widetilde{T}'$;
- $D \setminus z \stackrel{\text{def}}{=} (x_i : T_i - z)_{i:\, x_i \neq z}$;
- $D\,\sigma \stackrel{\text{def}}{=} \widetilde{x}\sigma : \widetilde{T}\sigma$.

We write $x\,D\,y$ *iff* $x \neq y$ *and* $x : T \in D$ *and* $y\ \mathcal{E}\ T$, *for some* T. *Given a substitution* σ *and a distinction* D, *we say that* σ respects D, *written* $\sigma \vdash D$, *if* $x\,D\,y$ *implies* $x\,\sigma \neq y\,\sigma$.

Definition 4 (structural congruence). *The structural congruence* $\equiv$ *is the least congruence on processes satisfying the abelian monoid laws for Summation and Composition (associativity, commutativity and* $\mathbf{0}$ *as identity) plus the following rules:*

$$\begin{aligned}
(\lambda x : T)\,(\lambda y : T' + x)\,P &\equiv (\lambda y : T')\,(\lambda x : T + y)\,P \quad && x \text{ not in } T' \\
(\lambda x : T)\,(\lambda y : T')\,P &\equiv (\lambda y : T')\,(\lambda x : T)\,P && x \not\mathcal{E}\ T' \\
(\lambda x)\,(P + Q) &\equiv (\lambda x)\,P\ +\ (\lambda x)\,Q \\
(\lambda x : T)\,\mathbf{0} &\equiv \mathbf{0} \\
!P &\equiv P \mid !P
\end{aligned}$$

and the scope extrusion *law:*

$$(\lambda x : T)\,(P|Q) \equiv (\lambda x : T)\,P|Q \quad x \notin \mathrm{fn}(Q) \text{ and } (T = \omega \text{ or } \omega \text{ does not occur in } Q).$$

Note that a special case of the first rule is: $(\lambda x)\,(\nu y)\,P \equiv (\nu y)\,(\lambda x : y)\,P$. The above structural congruence rules can be applied to reduce every pair of communicating subprocesses into a form where their ν-binders have been moved at top level. The side condition "ω does not occur in Q" prevents a λ-binder from 'capturing' ν-names out of its scope. For instance, according to the SOS rules presented below, a process $P = (\lambda x)\,ax \mid (\nu z)\,az$ can do a τ-transition. On the other side, extruding the scope of (λx) over $(\nu z)\,az$ would yield a process $P' = (\lambda x)\,(ax \mid (\nu z)\,az)$, which cannot do any synchronisation.

Definition 5 (labelled transition system). *The transition relation* $P \xrightarrow{\mu} Q$, *for* μ *a label of the form* $(\lambda\widetilde{y} : \widetilde{T})\,a\widetilde{v}$ (action) *or of the form* D, ϕ (effect) *is defined by the SOS rules in Table 1.*

Notation. In Table 1, we use the following abbreviation. We write $P > Q$ (and say that P *commits to* Q) if: $P = P_1 \,|\, P_2$, $P_1 \xrightarrow{(\lambda \widetilde{x}:\widetilde{T})\, a\widetilde{b}} P_1'$, $P_2 \xrightarrow{(\lambda \widetilde{y}:\widetilde{U})\, a\widetilde{c}} P_2'$, with ω not in $\widetilde{T}\widetilde{U}$, and $Q = (\lambda \widetilde{x}\widetilde{y} : \widetilde{T}\widetilde{U})\,(a\widetilde{b}.P_1' \,|\, a\widetilde{c}.P_2')$.

Some comments on the rules of Table 1 are in order. *Actions* occurring within the scope of a λ are governed by rules Pass and Open. Roughly, a name z that is declared with exceptions T' may get extruded (rule Open) or not (rule Pass) by an action occurring under the scope of its declaration $(\lambda z : T')$, depending on whether z occurs in the object part of the action. In the case of rule Pass, z is removed from the current set of exceptions $\widetilde{T}$. However, no distinction is lost, because the extruded names having z as an exception are added to T' (condition (1)). E.g.:

$$(\lambda z : a)\,(\lambda x : z)\, ax.P \xrightarrow{(\lambda x)\ ax} (\lambda z : \{a, x\})\, P.$$

Effects are similar to those found in Fusion, but here they also carry a set of exceptions represented by a distinction $\widetilde{x} : \widetilde{T}$. Effects are created as a result of a communication that unifies two tuples of names (rule Com), and propagated across parallel components, until a λ is encountered. The rule Pass_f has a meaning similar to Pass. The rule Open_f acts on a name z in the fusion: a substitutive effect $[w/z]$ is applied both to the transition label and to the target process, and z is removed from the fusion (the result is ϕ_{-z}). The side condition $\phi[z] \sqcap T'' = \emptyset$ forbids fusion of z with any name in its set of exceptions $(T' + \{T_j \,|\, z = x_j\})$, or having z as an exception $(\{x_i \,|\, z \mathcal{E} T_i\})$; in particular, the rule does not fire if $T' = \omega$, i.e. if z is declared to be new. Note that applying $[w/z]$ onto $(\widetilde{x}z : \widetilde{T}T')$ implicitly lets w inherit z's exceptions. For example:

$$(\lambda z : y)\,(\nu c)\,(cza.P | cww.\mathbf{0}) \xrightarrow{w:y,\,\{a=w\}} (\nu c)\, P[w/z]$$

while

$$(\lambda z : a)\,(\nu c)\,(cza.P | cww.\mathbf{0}) \not\rightarrow .$$

Here, z cannot be fused to w or to a, because a is in z's exceptions $(\phi[z] \sqcap T'' = \{a\})$. Note that in rule Com the labels in the premise have no binders because communication among processes with λ-binders is dealt by means of rule Commit. The absence of ν-binders in rule Commit is explained by the fact that, by the structural congruence, communicating processes can be reduced into a canonical form where the scope of ν-binders is extruded over parallel components.

Example 1.

1. The construct $(\lambda x)\, ax$ behaves as $a(x)$ in pi-calculus:

$$(\lambda x)\, ax.P \,|\, ay.\mathbf{0} \xrightarrow{\tau} P[y/x] \quad \text{and} \quad (\lambda x)\, ax.P \,|\, (\nu y)\, ay.\mathbf{0} \xrightarrow{\tau} (\nu y)\, P[y/x]$$
$$\text{but} \quad ax.P \,|\, (\lambda y : x)\, ay.\mathbf{0} \not\xrightarrow{\tau} \quad \text{and} \quad ax.P \,|\, (\nu y)\, ay.\mathbf{0} \not\xrightarrow{\tau}$$

Table 1. Actions and effects transitions in U-Calculus

$$\text{(Act)}\quad \widetilde{a\tilde{b}}.P \xrightarrow{a\tilde{b}} P \qquad \text{(Match)}\quad \frac{P \xrightarrow{\mu} Q}{[a=a]P \xrightarrow{\mu} Q} \qquad \text{(Sum)}\quad \frac{P \xrightarrow{\mu} Q}{P+R \xrightarrow{\mu} Q}$$

$$\text{(Pass)}\quad \frac{P \xrightarrow{(\lambda \widetilde{x}:\widetilde{T})\, a\tilde{b}} Q}{(\lambda z:T')\,P \xrightarrow{(\lambda \widetilde{x}:\widetilde{T}-z)\, a\tilde{b}} (\lambda z:T'')\;Q} \quad z \notin \widetilde{b},\, \widetilde{x} \cup \{a\} \text{ and (1)}$$

$$\text{(Open)}\quad \frac{P \xrightarrow{(\lambda \widetilde{x}:\widetilde{T})\, a\tilde{b}} Q}{(\lambda z:T')\,P \xrightarrow{(\lambda z \widetilde{x}:T'\widetilde{T})\, a\tilde{b}} Q} \quad z \in \widetilde{b} - \{a, \widetilde{x}\}$$

$$\text{(Pass}_f\text{)}\quad \frac{P \xrightarrow{\widetilde{x}:\widetilde{T},\phi} Q}{(\lambda z:T')\,P \xrightarrow{(\widetilde{x}:\widetilde{T})\setminus z,\phi} (\lambda z:T'')\,Q} \quad z \notin \mathrm{n}(\phi) \text{ and (1)}$$

$$\text{(Open}_f\text{)}\quad \frac{P \xrightarrow{\widetilde{x}:\widetilde{T},\phi} Q}{(\lambda z:T')\,P \xrightarrow{(\widetilde{x}z:\widetilde{T}T')[w/z],\phi_{-z}} Q[w/z]} \quad w\,\phi\,z,\ w \neq z,\ \phi[z] \sqcap T'' = \emptyset \text{ and (1)}$$

$$\text{(Com)}\quad \frac{P_1 \xrightarrow{a\tilde{b}} Q_1 \quad P_2 \xrightarrow{a\tilde{c}} Q_2}{P_1|P_2 \xrightarrow{\{\tilde{b}=\tilde{c}\}} Q_1|Q_2} \qquad \text{(Commit)}\quad \frac{P|Q > A \xrightarrow{\widetilde{x}:\widetilde{T},\phi} R}{P|Q \xrightarrow{\widetilde{x}:\widetilde{T},\phi} R}$$

$$\text{(Par)}\quad \frac{P \xrightarrow{\mu} Q}{P|R \xrightarrow{\mu} Q|R} \qquad \text{(Struct)}\quad \frac{P \equiv P' \quad P' \xrightarrow{\mu} Q' \quad Q' \equiv Q}{P \xrightarrow{\mu} Q}$$

(1) : Let $\widetilde{x} = x_1, \cdots, x_n$, and $\widetilde{T} = T_1, \cdots, T_n$ in

$T'' = T' + \Sigma\{x_i \mid z\ \mathcal{E}\ T_i\} + \{T_j \mid z = x_j\}$.

Symmetric rules for (Sum) and (Par) are not shown. Usual conventions about freshness of bound names apply.

2. The two examples below show combined use of rules (Com), (Open$_f$) and (Commit):

$$(\lambda x:y)\,(axx.P \mid awz.\mathbf{0}) \xrightarrow{w:y,\{w=z\}} P[w/x].$$
$$(\nu y)\,axyz.P \mid (\lambda x':z)\,(\lambda y')\,ax'y'z'.Q \xrightarrow{x:z,\{z=z'\}} (\nu y)\,(P \mid Q)[x/x'][y/y']$$

3. Nesting of binders is important, even on names in the same action:

$$(\nu y)\,(\lambda x)\,axy.\mathbf{0} \mid (\lambda u)\,auu \xrightarrow{\tau} \quad \text{while} \quad (\lambda x)\,(\nu y)\,axy.\mathbf{0} \mid (\lambda u)\,auu \not\xrightarrow{\tau}$$

Encoding I/O polarities We can encode polarities as follows:

$$\overline{a}\langle\widetilde{v}\rangle.P \triangleq (\nu x)\,(\lambda y)\,a\widetilde{v}xy.P \qquad a\langle\widetilde{v}\rangle.P \triangleq (\nu x)\,(\lambda y)\,a\widetilde{v}yx.P$$

for some chosen fresh x and y. The position of name x forbids fusions between actions with the same polarity and, hence, communication. For instance, the process $\overline{a}\langle\widetilde{v}\rangle.P|\overline{a}\langle\widetilde{u}\rangle.Q$ has no τ-transition, since the latter would force the fusion of two globally distinct names, which is forbidden by the operational rules. We denote by $\mathcal{U}^{\mathrm{P}}$, *polarised U-Calculus*, the subset of $\mathcal{U}$ in which every prefix can be interpreted as an input or output, in the above sense.

Open Bisimulation. Like in the case of Fusion, a 'natural' semantics of U-Calculus is required to be closed under substitutions. However, one should be careful in respecting exceptions raised by λ-extrusions. The following definition of *open bisimulation* relies on the notion of distinctions (Def. 3).

By $\{R_D\}_D$ we denote a set of process relations $\{R_D \mid D \text{ is a distinction}\}$. By $PR_{D'}Q$, with $D' = D \cdot (\nu\widetilde{x})\,,\ \widetilde{y} : \widetilde{T}$, we abbreviate $PR_{D''}Q$, with $D'' = D, x_1 : N_1, \cdots, x_k : N_k, \widetilde{y} : \widetilde{T}$ and $N_i = \mathrm{fn}(P, Q, D,) \cup \{x_1, \cdots, x_{i-1}\}$, with $i = 1, \cdots k$.

Definition 6 (open bisimulation). *A set $\mathcal{R} = \{R_D\}_D$ of process relations indexed by distinctions is an* indexed simulation *if for each D, whenever $P\ R_D\ Q$:*

- *if $P \xrightarrow{(\nu x)\,(\lambda\widetilde{y}:\widetilde{T})\,a\widetilde{z}} P'$ then $Q \xrightarrow{(\nu x)\,(\lambda\widetilde{y}:\widetilde{T})\,a\widetilde{z}} Q'$ and $P'\ R_{D'}\ Q'$, with*
$$D' = D \cdot (\nu\widetilde{x})\,,\ \widetilde{y} : \widetilde{T};$$
- *if $P \xrightarrow{\widetilde{x}:\widetilde{T},\phi} P'$, σ is a substitutive effect of ϕ and σ respects $D, \widetilde{x} : \widetilde{T}$ then $Q \xrightarrow{\widetilde{x}:\widetilde{T},\phi} Q'$ and $P'\sigma\ R_{D''}\ Q'\sigma$, with*
$$D'' = \big(D,\ \widetilde{x} : \widetilde{T}\,\big)\sigma.$$

$\mathcal{R}$ is an indexed bisimulation *if both $\mathcal{R} = \{R_D\}_D$ and $\mathcal{R}^{-1} = \{R_D^{-1}\}_D$ are indexed simulations.* Open bisimulation, *$\{\sim_D\}_D$, is the largest indexed bisimulation preserved by respectful substitutions, i.e.: for each σ and distinction D, if $P \sim_D Q$ and σ respects D then $P\sigma \sim_{D\sigma} Q\sigma$.*

We write $P \sim Q$ for $P \sim_\epsilon Q$, where ϵ is the empty distinction. In the following examples we shall write $\{\widetilde{x} = \widetilde{y}\}.P$ for $(\nu c)\,(c\widetilde{x}|c\widetilde{y}.P)$ (for a fresh name c).

Example 2.

1. $(\nu c)\,(\nu n)\,(\lambda x)\,\big(cx.P \mid cn.\mathbf{0}\big) \sim (\nu c)\,\big((\lambda x)\,cx.P \mid (\nu n)\,cn.\mathbf{0}\big) \sim (\nu c)\,(\nu n)\,\tau.P[^{n}/_{x}]$
but $(\nu c)\,(\lambda x)\,(\nu n)\,\big(cx.P \mid cn.\mathbf{0}\big) \sim (\nu c)\,(\lambda x)\,\big(cx.P \mid (\nu n)\,cn.\mathbf{0}\big) \sim \mathbf{0}$.
2. An example of 'expansion' for parallel composition is as follows:
$$(\lambda y : T)\,ay.\mathbf{0}|ax.\mathbf{0} \ \sim\ (\lambda y : T)\,ay.ax.\mathbf{0} \ +\ ax.(\lambda y : T)\,ay.\mathbf{0} \ +\{x = y\}.\mathbf{0} \quad \text{if } x \notin T,$$
$$\text{while } (\lambda y : T)\,ay.\mathbf{0}|ax.\mathbf{0} \ \sim\ (\lambda y : T)\,ay.ax.\mathbf{0} \ +\ ax(\lambda y : T)\,ay.\mathbf{0} \quad \text{if } x\ \mathcal{E}\ T$$

3. The static nesting of name binding is relevant:

$$(\nu y)\,(\lambda x)\,ay.ax.\{x = y\}.\mathbf{0} \not\sim (\lambda x)\,(\nu y)\,ay.ax.\{x = y\}.\mathbf{0}.$$

The above two processes extrude y and x in the same order. However, after the two extrusions, the process on the left-hand side can fuse the two names, while the other one cannot.

Theorem 1. *Let P and Q be two processes. Then:*

1. *$P \sim_{D,x:T\sqcap N} Q$ and $x \notin \mathrm{n}(D)$ imply $(\lambda x : T)\,P \sim_D (\lambda x : T)\,Q$, with $N = \mathrm{fn}(\,(\lambda x : T)\,P,\,(\lambda x : T)\,Q,\,D\,)$.*
2. *Prefix, parallel composition, sum, matching and replication operators preserve $\sim_D$.*

Example 3. Let $P \stackrel{\text{def}}{=} (\lambda y)\,ay.(\nu x)\,ax.\mathbf{0}$ and $Q \stackrel{\text{def}}{=} (\lambda y)\,ay.(\nu x)\,ax.\{x = y\}.Q'$. It holds that $P \sim Q$. Indeed, Q cannot fuse x and y, since ν-extruding x yields a distinction $x : y$. Suppose $R \stackrel{\text{def}}{=} (\lambda z)\,az$. It also holds that $P \mid R \sim Q \mid R$. Indeed, after synchronising $(\lambda z)\,az$ and $(\lambda y)\,ay$, $Q \mid R$ ν-extrudes x and then evolves, for instance, to $(\lambda z : x)\,\{x = z\}.Q'[z/\!\!/y]$. Thus, the fusion $\{x = z\}$ cannot take place and $(\lambda z : x)\,\{x = z\}.Q'[z/\!\!/y] \sim \mathbf{0}$.

3 Pi-Calculus and Fusion as Subcalculi of U-Calculus

The labelled transition systems of pi-calculus and Fusion are embedded into polarised U-calculus's, under the two obvious translations given below. Note that these translations are *uniform*, in the sense of [1]; in particular, no central coordinator is introduced in the translated processes.

Definition 7. *The translations $[\![\cdot]\!]_\pi : \Pi \to \mathcal{U}^{\mathrm{p}}$ and $[\![\cdot]\!]_f : \mathcal{F} \to \mathcal{U}^{\mathrm{p}}$ are defined by extending in the expected homomorphic way the following clauses, respectively:*

$$[\![\overline{a}\langle x\rangle.P]\!]_\pi = \overline{a}\langle x\rangle.[\![P]\!]_\pi \quad [\![a(x).P]\!]_\pi = (\lambda x)\; a\langle x\rangle.[\![P]\!]_\pi \quad [\![(\nu x)\,P]\!]_\pi = (\nu x)\,[\![P]\!]_\pi$$

$$[\![\overline{a}\langle x\rangle.P]\!]_f = \overline{a}\langle x\rangle.[\![P]\!]_f \qquad [\![a\langle x\rangle.P]\!]_f = a\langle x\rangle.[\![P]\!]_f \qquad [\![(x)P]\!]_f = (\lambda x)\;[\![P]\!]_f$$

Embedding in terms of labelled transition systems naturally lifts to behavioural equivalences. Here, we restrict our attention to equivalences based on barbed bisimulation.

Definition 8 (barbed bisimulation and barbed congruence). *We write $P \downarrow a$ if and only if there exist an action $\mu = ((\lambda \widetilde{x} : \widetilde{T})\,, a\widetilde{v})$ and a process Q such that $P \xrightarrow{\mu} Q$.*

A barbed bisimulation *is a symmetric binary relation $\mathcal{R}$ between processes such that $P\,\mathcal{R}\,Q$ implies:*

1. *whenever* $P \xrightarrow{\tau} P'$ *then* $Q \xrightarrow{\tau} Q'$ *and* $P' \mathcal{R} Q'$*;*
2. *for each name* a*, if* $P \downarrow a$ *then* $Q \downarrow a$*.*

P is barbed bisimilar to Q, written $P \dot{\sim} Q$, if $P \mathcal{R} Q$ for some barbed bisimulation $\mathcal{R}$.

Two processes P and Q are barbed congruent*, written $P \sim^b Q$, if for all contexts $C[\cdot]$, it holds that $C[P] \dot{\sim} C[Q]$.*

Let $\sim^\pi$ and $\sim^{\mathrm{f}}$ denote barbed congruence, respectively, over Π ([12]) and over $\mathcal{F}$ (see [14]). Also, let $\sim^{[\![\pi]\!]}$ and $\sim^{[\![\mathrm{f}]\!]}$ be the equivalences on $\mathcal{U}$ obtained by closing barbed bisimulation $\dot{\sim}$ only under translated pi- and Fusion-contexts, respectively (e.g., $P \sim^{[\![\pi]\!]} Q$ iff for each Π-context $C[\cdot]$, $[\![C]\!]_\pi[P] \dot{\sim} [\![C]\!]_\pi[Q]$).

Proposition 1.

1. *Let P and Q be two pi-calculus processes. $P \sim^\pi Q$ iff $[\![P]\!]_\pi \sim^{[\![\pi]\!]} [\![Q]\!]_\pi$.*
2. *Let P and Q be two Fusion processes. $P \sim^{\mathrm{f}} Q$ iff $[\![P]\!]_f \sim^{[\![\mathrm{f}]\!]} [\![Q]\!]_f$.*

Next, we now show that the U-calculus cannot be uniformly encoded into Π. The intuition is that, in U-calculus (like in D-Fusion [1]), the combined use of fusions and restrictions allows one to express a pattern matching atomically. This is not possible in Π. To show this fact, we restrict our attention to polarised U-calculus, $\mathcal{U}^{\mathrm{P}}$.

The reference semantics for Π is the late operational semantics. Given $P \in \Pi$ and a trace of U-calculus actions s, let us write $P \stackrel{\hat{s}}{\Longrightarrow}$ if $P \stackrel{s'}{\Longrightarrow}$ for some pi-actions trace s' that exhibits the same sequence of subject names as s, with the same polarities (e.g., $s = a\langle\widetilde{x}\rangle \cdot (\lambda\widetilde{y})\, \overline{b}\langle\widetilde{v}\rangle$ and $s' = a(\widetilde{z}) \cdot \overline{b}\langle\widetilde{w}\rangle$). The reference semantics for Π is again the late operational semantics.

Definition 9. *A translation $[\![\cdot]\!] : \mathcal{U}^{\mathrm{P}} \to \Pi$ is* uniform *if for each $P, Q \in \mathcal{U}^{\mathrm{P}}$:*

- *for each trace s, $P \stackrel{s}{\Longrightarrow}$ implies $[\![P]\!] \stackrel{\hat{s}}{\Longrightarrow}$;*
- *$[\![P|Q]\!] = [\![P]\!]|[\![Q]\!]$;*
- *for each y, $[\![(\nu y)\, P]\!] = (\nu y)\, [\![P]\!]$;*
- *for each substitution σ, $[\![P\sigma]\!] = [\![P]\!]\sigma$.*

Below, we denote by $\sim_{\mathcal{U}^{\mathrm{P}}}$ any fixed equivalence over $\mathcal{U}^{\mathrm{P}}$ contained in trace semantics (defined in the obvious way), and by $\sim_\Pi$ any fixed equivalence over Π contained in trace equivalence. Note that both barbed congruence over $\mathcal{U}^{\mathrm{P}}$, and open bisimulation are contained in trace equivalence.

Proposition 2. *There is no uniform translation $[\![\cdot]\!] : \mathcal{U}^{\mathrm{P}} \to \Pi$ such that $\forall\, P, Q \in \mathcal{U}^{\mathrm{P}}$:*

$$P \sim_{\mathcal{U}^{\mathrm{P}}} Q \Rightarrow [\![P]\!] \sim_\Pi [\![Q]\!].$$

Proof: Suppose that there exists such a translation $[\![\cdot]\!]$. Let us consider the following two $\mathcal{U}^{\mathrm{P}}$-processes P and Q:

$$P = (\nu c, k, h)\, (c\langle k\rangle.\overline{a}.\mathbf{0} | c\langle h\rangle.\overline{b}.\mathbf{0} | \overline{c}\langle k\rangle.\mathbf{0}) \qquad Q = \tau.\overline{a}.\mathbf{0}.$$

It holds that $P \sim Q$ in $\mathcal{U}^{\mathrm{P}}$: the reason is that, in P, synchronisation between prefixes $c\langle h\rangle$ and $\overline{c}\langle k\rangle$, which carry different *restricted* names h and k, is forbidden (see rule $\textsc{Pass}_f$). Thus P can only make $c\langle k\rangle$ and $\overline{c}\langle k\rangle$ synchronise, and then perform $\overline{a}$. Thus, $P \sim_{\mathcal{U}^{\mathrm{P}}} Q$ holds too.

On the other hand, by Definition 9, for any uniform encoding $[\![\cdot]\!]$, c and $\bar{c}$ in $[\![P]\!]$ can synchronise and, thus, $[\![P]\!] \stackrel{\overline{b}}{\Longrightarrow}$, while $[\![Q]\!] \stackrel{\overline{b}}{\not\Longrightarrow}$ (because of $b \notin \mathrm{fn}(Q)$ and of the uniformity with respect to substitutions). Thus $[\![P]\!] \not\sim_{\Pi} [\![Q]\!]$. □

Of course, it is also true that the U-calculus cannot be uniformly encoded into $\mathcal{F}$, as this would imply the existence of a uniform fully abstract encoding from Π to $\mathcal{F}$, which does not exist (see [1]).

The conclusion is that there is some expressiveness gap between U-calculus on one side and Pi/Fusion on the other side, at least, as far as our simple notion of uniform encoding is concerned.

Remark. There cannot exist any encoding from D-Fusion to the U-calculus, or vice-versa, that is uniform in a sense extending Def. 9, in particular mapping λ to λ and ν to ν. The reason is that in D-Fusion, as mentioned, the order of λ's and ν's can be freely swapped, while in the U-calculus this requires changing the respective exceptions. More in detail, the equality $(\lambda x)\,(\nu n)\,\{x = n\}.\bar{c} \sim \mathbf{0}$ in the U-calculus would be mapped to $[\![(\lambda x)\,(\nu n)\,\{x = n\}.\bar{c}]\!] \sim_{\mathcal{DF}} [\![\mathbf{0}]\!]$ in D-Fusion (for $\sim_{\mathcal{DF}}$ included in trace equivalence). In D-Fusion, using commutativity of ν and λ, one would get $[\![(\nu n)\,(\lambda x)\,\{x = n\}.\bar{c}]\!] = [\![P]\!] \sim_{\mathcal{DF}} [\![\mathbf{0}]\!]$. But this equivalence does not hold true, since $P \stackrel{\bar{c}}{\Longrightarrow}$ implies by definition that $[\![P]\!] \stackrel{\bar{c}}{\Longrightarrow}$, while $[\![\mathbf{0}]\!] \stackrel{\bar{c}}{\not\Longrightarrow}$ (the latter follows by uniformity with respect to substitutions). This shows that the U-calculus cannot be uniformly encoded into D-Fusion. A similar argument applies to the other direction (that is, mapping D-Fusion to U-Calculus).

4 Encoding Guarded Choice

We show that in U-calculus, like in D-Fusion [1], the combined use of fusions and restrictions can still be used to uniformly encode guarded mixed choice *via* parallel composition. Practically, this guarantees that there is no significant loss of expressive power when moving from D-Fusion to U-calculus.

In the encoding, different branches of a guarded choice will be represented as concurrent processes. The encodings add pairs of extra names to the object part of each action: these extra names are used as 'side-channels' for atomic coordination among the different branches. Let us first look at a simple example.

Example 4. Consider the guarded choice $A = (\nu n)\,(\lambda x)\,a\langle xn\rangle.P + (\nu m)\,(\lambda x)\,a\langle xm\rangle.Q$. Its intended 'parallel' implementation is the process:

$$B = (\nu n)\,(\nu m)\,(\lambda x)\,\big(\, a\langle xn\rangle.P \mid a\langle xm\rangle.Q \,\big)$$

(here, $x, n, m \notin \mathrm{fn}(a, P, Q)$). Assume parallel contexts are constrained so that output actions on channel a must carry two identical names. In B, the parallel

component that first consumes any such message, forces fusion of x either to n or to m, and consequently inhibits the other component, thus:

$$(\lambda u)\,\overline{a}\langle uu\rangle | B \xrightarrow{\tau} \sim\ (\nu n)\,(P \mid (\nu m)\,a\langle mn\rangle.Q) \quad \sim \quad P \mid (\nu n, m)\,a\langle mn\rangle.Q.$$

Under the mentioned assumption, $(\nu m, n)\,a\langle mn\rangle.Q$ should be 'equivalent' to $\mathbf{0}$, because there is no way of fusing m and n together. In other words, choice between P and Q has been resolved atomically. Note that this example exploits in a crucial way features of both Fusion (sharing of the variable x, in B) and of U-calculus (restricted input).

We generalise the above example by providing a fully abstract encoding of mixed guarded choice. For the sake of simplicity, we shall work here with barbed equivalence. We believe the results can also be stated in terms of labelled bisimilarity $\sim$, at the cost of breaking uniformity of the encoding (e.g. by introducing of 'firewalls' contexts which filter out output messages that disrupt the encoding, see [1]) .

As a source language we fix a sorted version of polyadic pi-calculus [5] with 'mixed' choice, Π^{mix}. In this language, prefixes and + are replaced by mixed summation, $\sum_{i\in I} a_i\,(\widetilde{x}_i).P_i + \sum_{j\in J} \overline{b_j}\langle\widetilde{v}_j\rangle.Q_j$. The target language is the fragment of polarised U-Calculus with no summation at all. The relevant clause is shown below, where $\widetilde{n} = (n_i)_{i\in I}$ and $\widetilde{m} = (m_j)_{j\in J}$ are two disjoint tuples of distinct names:

$$\begin{array}{l}[\![\sum_{i\in I} a_i\,(\widetilde{x}_i).P_i + \sum_{j\in J} \overline{b_j}\langle v_j\rangle.Q_j]\!]_{\mathrm{mix}} \quad = \\ (\nu\widetilde{n}\widetilde{m})\,((\lambda z, u)\,)\,(\ \Pi_{i\in I}(\lambda\widetilde{x}_i)\,a_i\langle\widetilde{x}_i z n_i uu\rangle.[\![P_i]\!]_{\mathrm{mix}}\ \mid\ \Pi_{j\in J}\overline{b_j}\langle\widetilde{v}_j uuz m_j\rangle.[\![Q_j]\!]_{\mathrm{mix}}\ ).\end{array}$$

The encoding acts as a homomorphism over the remaining operators of Π^{mix}. Note that, differently from [1], the declaration of the λ-names is within the scope of the ν-names. Communication between two remote prefixes of opposite polarities causes all λ-names within the same choice to be fused to a single ν-name. This atomically inhibits the remaining prefixes. Note that the relative positions of ν-names correctly forbid communication between branches of opposite polarities within the same choice (no 'incestuous' communication, according to the terminology of [7]).

Below, $\sim^{\mathrm{mix}}$ denotes barbed congruence over Π^{mix}, and $\sim^{[\![\mathrm{mix}]\!]}$ the equivalence over the U-calculus obtained by closing barbed bisimulation under translated Π^{mix}-contexts, i.e.: $P \sim^{[\![\mathrm{mix}]\!]} Q$ iff for each Π^{mix}-context $C[\cdot]$, it holds $[\![C]\!]_{\mathrm{mix}}[P] \dot{\sim} [\![C]\!]_{\mathrm{mix}}[Q]$). Both equivalences are *reasonable* semantics in the sense of [9]. The proof of the following theorem is straightforward, given that there is a 1-to-1 correspondence between reductions and barbs of R and of $[\![R]\!]_{\mathrm{mix}}$, for any R, and given that the encoding is compositional, in particular, for any context $C[\cdot]$, it holds $[\![C]\!]_{\mathrm{mix}}[[\![P]\!]_{\mathrm{mix}}] = [\![C[P]]\!]_{\mathrm{mix}}$.

Theorem 2 (full abstraction for mixed choice). *Let $P, Q \in \Pi^{\mathrm{mix}}$. It holds that $P \sim^{\mathrm{mix}} Q$ if and only if $[\![P]\!]_{\mathrm{mix}} \sim^{[\![\mathrm{mix}]\!]} [\![Q]\!]_{\mathrm{mix}}$.*

In a pi-calculus setting, it is well-known that mixed choice cannot be encoded into the choice-free fragment, if one requires the encoding be uniform and preserve a reasonable semantics [8,9,7]. The theorem above shows that pi-calculus mixed choice *can* be implemented into the choice-free fragment of the U-calculus. The encoding is uniform, deadlock- and divergence-free, and preserves a reasonable semantics.

5 Conclusions

We have introduced U-Calculus, a process calculus with no I/O polarities and a unique binding, that can be used both to control the scope of fusions and new name generation. This is achieved by means of a simple form of typing that prevents a name x such that $x : T$ from being fused with any name in T.

We have proved that the U-Calculus is strictly more expressive than pi-calculus and Fusion calculus separately. Remarkably, thanks to the combination of static and dynamic ordering of names, the labelled bisimulation defined for the U-Calculus is a congruence. This property represents a substantial improvement with respect to D-Fusion.

We plan to extend the U-Calculus by generalising name fusions to substitutions over an arbitrary signature of terms. We believe that the extended calculus would be strictly more expressive that Logic Programming, the intuition being that restriction (creation of new fresh names) cannot be modelled in LP.

It would also be interesting to investigate whether the partition refinement algorithm proposed in [11] for checking open bisimilarity could be extended to U-Calculus.

References

1. M. Boreale, M. Buscemi, U. Montanari. D-Fusion: a Distinctive Fusion Calculus. In Proc. of APLAS'04, LNCS 3302, Springer-Verlag, 2004.
2. P. Gardner, C. Laneve, and L. Wischik. The fusion machine (extended abstract). In *Proc. of CONCUR '02*, LNCS 2421. Springer-Verlag, 2002.
3. L. G. Meredith, S. Bjorg, and D. Richter. Highwire Language Specification Version 1.0. Unpublished manuscript.
4. Microsoft Corp. Biztalk Server - `http://www.microsoft.com/biztalk`.
5. R. Milner. The Polyadic pi-Calculus: a Tutorial. Technical Report, Computer Science Dept., University of Edinburgh, 1991.
6. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.
7. U. Nestmann and B. C. Pierce. Decoding choice encodings. *Information and Computation*, 163(1):1–59, 2000.
8. C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous pi-calculus. In *Conf. Rec. of POPL'97*, 1997.
9. C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous pi-calculus. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

10. J. Parrow and B. Victor. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. In *Proc. of LICS'98*. IEEE Computer Society Press, 1998.
11. M. Pistore and D. Sangiorgi. A Partition Refinement Algorithm for the Pi-Calculus. *Information and Computation*, 164(2): 264–321, 2001.
12. D. Sangiorgi. Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.
13. D. Sangiorgi. A Theory of Bisimulation for the pi-Calculus. *Acta Informatica*, 33(1): 69-97, 1996.
14. B. Victor. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. PhD thesis, Department of Computer Systems, Uppsala University, 1998.