Concurrent Constraints in the Fusion Calculus
(extended abstract)

Björn Victor[1] and Joachim Parrow[2]

[1] Dept. of Computer Systems, Uppsala University, Sweden, Bjorn.Victor@DoCS.UU.SE
[2] Dept. of Teleinformatics, Royal Institute of Technology, Sweden, joachim@it.kth.se

Abstract. We use the fusion calculus, a generalization and simplification of the π-calculus, to model concurrent constraint programming. In particular we encode three basic variants of the ρ-calculus, which is a foundational calculus for the concurrent constraint programming language Oz. Using a new reduction-based semantics and weak barbed congruences for the fusion calculus we formally establish an operational correspondence between the ρ-calculi and their encodings. These barbed congruences are shown to coincide with the hyper-equivalences previously adopted for the fusion calculus.

1 Introduction

In this paper we use the fusion calculus to model concurrent constraint programming, thereby relating the paradigm of communicating processes to that of concurrent constraints. In the first, parallel agents interact with each other by sending and receiving data over named ports; in the second, agents produce constraints on the values of variables, which are combined to resolve queries about the values. Previous attempts at reconciling these paradigms have used one of two different lines: either extending a process calculus with logical variables [12] or encoding the variables as separate processes [15]. In the present paper our method is more direct, highlighting the fact that the fusion calculus serves as a basic underlying formalism. To establish correctness we explore the reduction semantics and weak, or observation, equivalences of the fusion calculus in detail.

Process Calculi: The fusion calculus is a calculus for mobile processes, in the sense that the communication structure of the involved processes may change over the execution of the system. The π-calculus [4] is the standard example of such a calculus. Here, the data sent and received over named ports are port names; there is no difference between ports and data, and they are collectively called *names*. An example of interaction in the π-calculus is:

$$\pi\text{-calculus:} \qquad \overline{u}\langle x\rangle \,.\, P \mid u(y)\,.\,Q$$

which can evolve into $P \mid Q\{x/y\}$ through communication over the name u.

The expressive power of mobile processes has been established by using the π-calculus for semantics of programming languages such as PICT [9] and of object-oriented programming [16], for encoding higher-order communications [10] and the λ-calculus [3]. Sequential logic programming [2] and simple concurrent constraint programming [15] have also been modelled: in both these cases, the major obstacle has been expressing logical variables, which can be updated from several processes and not only at their binding occurrence. The variables are modelled by separate processes which can be read, updated etc. by sending requests and receiving responses.

In a previous paper [7] we show how the π-calculus can be *simplified* into the fusion calculus while at the same time extending its expressiveness. The heart of the fusion calculus is the *fusion action*, which in interplay with the *scope* operator can express changes to names shared between several processes in the system. The changes amount to identifying, or *fusing*, previously distinct names. A sample interaction is:

$$\text{fusion calculus:} \qquad (y)(\overline{u}x\,.\,P \mid uy\,.\,Q \mid R)$$

which can evolve into $(P \mid Q \mid R)\{x/y\}$ by communication over the name u. In this example the communication has effect not only in Q, but in all the agents in the scope of y. Although the fusion calculus is a simplification, it gains considerable expressive power over the π-calculus, and we show in this paper that it is easy to model logical variables in a few basic constraint systems.

Concurrent Constraints: In concurrent constraints, the effect of computation consists of changes to a shared *constraint store* by adding variable constraints to it by *tell* operations. The store combines the constraints, and can subsequently be queried by *ask* operations to find out if a formula containing constrained variables is entailed. The store differs from a normal memory store, which holds *values* (such as 42) of variables; here a variable can have a *partial* value (such as "between 17 and x"). An example of a concurrent constraint computation is

$$\text{CC-program:} \qquad \mathit{ask}(x < y)\,.\,P \wedge \mathit{tell}(y > 17) \wedge \mathit{tell}(x < 11)$$

which can evolve into P with a store which entails $x < 11 \wedge y > 17$.

To relate mobile process calculi and concurrent constraints, we use the *ρ-calculus* of Niehren and Müller [6]. This is a calculus which serves as a foundation of the concurrent constraint programming language Oz, developed at DFKI in Saarbrücken. Oz incorporates constraints, functional, and object-oriented programming in a concurrent setting, and requires a richer foundational calculus than a pure constraint calculus as in, e.g., [11]. The process part of the ρ-calculus is a subcalculus of the π-calculus [6], and it is expressive enough to model functional and object-oriented programming. It can be extended by an arbitrary constraint system $\mathcal{CS}$, resulting in the calculus $\rho(\mathcal{CS})$. This takes the calculus beyond traditional process calculi and it thus cannot directly use the rich flora of associated theories and tools. An example of a term in $\rho(x = y, x \neq y)$, with constraints being equalities or inequalities, is

$$\rho(x = y, x \neq y): \quad x = u \ \wedge\ u \neq y \ \wedge\ \texttt{if}\ x = y\ \texttt{then}\ E_1\ \texttt{else}\ (z = w \wedge E_2)$$

which can evolve into $x = u \ \wedge\ u \neq y \ \wedge\ z = w \ \wedge\ E_2$. The top-level equality and inequality conjuncts in this way represent the store and the `if`-conjunct corresponds to *ask*.

The main technical point of this paper is to represent $\rho(\mathcal{CS})$ for a few basic $\mathcal{CS}$ in the fusion calculus. The semantics of the ρ-calculus is given in [6] using reduction rules and a structural congruence resulting in an *unlabelled* transition system. (Term-rewriting systems such as the λ-calculus also have reduction semantics.) The semantics of the fusion calculus, on the other hand, is given in [7] using rules for *labelled* transitions (and a similar structural congruence), as traditionally has been done for process calculi such as CCS or the π-calculus. To relate the calculi we give a reduction semantics for the fusion calculus. The behaviour of an agent must then be considered together with an environment with which it can interact. We verify that the reduction semantics corresponds with the old labelled transition semantics, by defining appropriate equivalences between agents in the new semantics, and showing that it coincides with the equivalences of the old semantics.

Main Results and Overview: In the next Section, we review from [7] the syntax and labelled transition semantics of the fusion calculus, and the definition of (strong) hyperequivalence. We add a definition of *weak* hyperequivalence, and show that in the fusion calculus it is the weakest congruence which contains weak bisimulation. In Section 3 we give a reduction semantics to the fusion calculus, define strong and weak congruences in this semantics, based on *barbed bisimulation* [5, 10], and verify that they coincide with strong and weak hyperequivalence. In Section 4 we present the syntax and semantics of the ρ-calculus and show how to encode it in the fusion calculus for its instantiation to three basic constraint systems. We show that the encodings are correct by verifying an operational correspondence between reductions in the ρ-calculi and in the encodings. Finally in Section 5 we conclude and give some directions for future work.

Related work: The separation of binding and update in process calculi was, to our knowledge, first done independently by Fu [1] with his χ-calculus and by us [8] with the update calculus. The update calculus turned out not to generalize to a polyadic calculus (where more than one name at a time can be transmitted), and therefore we developed it further into the fusion calculus [7].

The Oz programming language is not only a concurrent constraint programming language, but also supports higher-order functional programming and concurrent objects, and requires a richer foundational calculus than a pure constraint calculus as in, e.g., [11]. The ρ-calculus [6] is more appropriate in this respect.

In [12], Smolka defined the γ-calculus and informally related it to a variant of the π-calculus with logical variables, equational constraints and a constraint elimination rule. Niehren and Müller in [6] defined the ρ-calculus, and related $\rho(\emptyset)$ (the ρ-calculus without constraints) to a subcalculus of the π-calculus, and $\rho(x = y)$ to a variant of the γ-calculus without constants. In [15], we related the π-calculus directly to the γ-calculus by an encoding which we proved correct. In the present paper we strengthen these results by relating the fusion calculus to both $\rho(x = y)$ and $\rho(x = y, \mathcal{C})$. (In [7] it is shown that the π-calculus is a subcalculus of the fusion calculus.) Compared to the encodings in this paper, the encoding of [15] was quite complex due to the difficulties of representing logical variables. Logical variables as part of Prolog were specified in [2], but no proofs of correctness were presented.

The theory of barbed bisimulation originated in [5], where CCS was treated, and was further developed in [10] for the π-calculus and the higher-order π-calculus.

2 The Fusion Calculus

In this section we first review the syntax and operational semantics of a version of the fusion calculus with match, mismatch and recursion. We then review the strong bisimulation equivalence and congruence from [7]. The novel contribution of the present paper begins by exploring the weak bisimulation equivalences. We show that the weak congruence, *weak hyperequivalence*, is the only reasonable congruence in the sense that it is the largest congruence contained in weak bisimulation.

2.1 Syntax and Semantics

We assume an infinite set $\mathcal{N}$ of *names* ranged over by $u, v, \ldots, z$. Like in the π-calculus, names represent communication channels, which are also the values transmitted. We write $\tilde{x}$ for a (possibly empty) finite sequence $x_1 \cdots x_n$ of names. φ ranges over total equivalence relations over $\mathcal{N}$ (i.e. equivalence relations with $\mathrm{dom}(\varphi) = \mathcal{N}$) with only finitely many non-singular equivalence classes. We write $\{\tilde{x} = \tilde{y}\}$ to mean the smallest such equivalence relation relating each x_i with y_i, and write $\mathbf{1}$ for the identity relation.

Definition 1. *The* free actions, *ranged over by α, and the* agents, *ranged over by $P, Q, \ldots$, are defined by*

$$\begin{array}{lll} \alpha ::= u\tilde{x} & \text{(Input)} & \quad P ::= \mathbf{0} \;\mid\; \alpha \,.\, Q \;\mid\; Q + R \;\mid\; Q \mid R \;\mid \\ \quad\;\; \overline{u}\tilde{x} & \text{(Output)} & \qquad\quad (x)Q \;\mid\; [x = y]Q \;\mid\; [x \neq y]Q \;\mid\; A(\tilde{x}) \\ \quad\;\; \varphi & \text{(Fusion)} & \end{array}$$

The input and output actions above are collectively called *free communication* actions. In these, the names $\tilde{x}$ are the *objects* of the action, and the name u is the *subject*. We write a to stand for either u or $\overline{u}$, thus $a\tilde{x}$ is the general form of a communication action. We identify fusion actions not based on their concrete syntax, but based on their corresponding equivalence relations. Fusion actions have neither subject nor objects.

Prefixing an agent Q means that the agent must perform the action α before acting like Q. Performing a fusion here represents an obligation to treat all related names as identical. We often omit a trailing $\mathbf{0}$ and write α for $\alpha \,.\, \mathbf{0}$ if no confusion can arise. Like in the π-calculus, Summation $Q + R$ is alternative choice and Composition $Q \mid R$ lets agents act in parallel.

The Scope $(x)Q$ limits the scope of x to Q; no visible communication action of $(x)Q$ can have x as its subject, and fusion effects with respect to x are limited to Q. Restriction and input binding of the π-calculus can be recovered as special cases of Scope.

A Match $[x = y]Q$ acts like Q if x and y are the same name; a Mismatch $[x \neq y]Q$ acts like Q if x and y are *not* the same. We let M, N range over Match or Mismatch operators and A over a possibly infinite set of identifiers, each with an associated nonnegative arity k and agent definition $A(x_1, \ldots, x_k) \stackrel{def}{=} P$ with pairwise distinct x_i such that $\mathrm{fn}(P) \subseteq \{x_1, \ldots, x_k\}$.

The name x is said to be *bound* in $(x)P$. We write $(\tilde{x})P$ for $(x_1) \cdots (x_n)P$. The *free names* in P, denoted $\mathrm{fn}(P)$, are the names in P with a non-bound occurrence; here the names occurring in the fusion φ is defined to be the names in the non-singular equivalence classes, i.e. in the relation $\varphi - \mathbf{1}$. As usual we will not distinguish between alpha-variants of agents, i.e., agents differing only in the choice of bound names.

The action of a transition may be free or bound:

Definition 2. *The* actions, *ranged over by γ, consist of the fusion actions and of communication actions of the form $(z_1) \cdots (z_n)a\tilde{x}$ (written $(\tilde{z})a\tilde{x}$), where $n \geq 0$ and all elements in $\tilde{z}$ are also in $\tilde{x}$. If $n > 0$ we say it is a* bound *action.*

Note that there are no bound fusion actions. In the communication actions above, $\tilde{z}$ are the *bound objects* and the elements in $\tilde{x}$ that are not in $\tilde{z}$ are the *free objects*. We further write $\mathrm{n}(\gamma)$ to mean all names occurring in γ (i.e., also including the subject of communication actions and the names in non-singular equivalence classes in fusion actions).

For convenience we define $\varphi\backslash z$ to mean $\varphi \cap (\mathcal{N} - \{z\})^2 \cup \{(z,z)\}$, i.e., the equivalence relation φ with all references to z removed (except for the identity). For example, $\{x=y\}\backslash y = \mathbf{1}$, and $\{x=z, z=y\}\backslash z = \{x=y\}$.

We now define a structural congruence which equates all agents we will never want to distinguish for any semantic reason, and then use this when giving the transition semantics.

Definition 3. *The* structural congruence, $\equiv$, *between agents is the least congruence satisfying the abelian monoid laws for Summation and Composition (associativity, commutativity and* $\mathbf{0}$ *as identity), and the scope laws* $(x)\mathbf{0} \equiv \mathbf{0}$, $\quad (x)(y)P \equiv (y)(x)P, (x)(P+Q) \equiv (x)P + (x)Q, \quad (x)MP \equiv M(x)P$ *if* $x \notin \mathrm{n}(M)$ *and also the* scope extension *law* $P \mid (z)Q \equiv (z)(P \mid Q)$ *where* $z \notin \mathrm{fn}(P)$*, and the law for process identifiers:* $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ *if* $A(\tilde{x}) \stackrel{def}{=} P$.

Definition 4. *The family of* transitions $P \xrightarrow{\gamma} Q$ *is the least family satisfying the laws in Table 1. In this definition structurally equivalent agents are considered the same, i.e., if* $P \equiv P'$ *and* $Q \equiv Q'$ *and* $P \xrightarrow{\gamma} Q$ *then also* $P' \xrightarrow{\gamma} Q'$.

$$\textsc{pref}\ \frac{-}{\alpha\,.\,P \xrightarrow{\alpha} P} \qquad \textsc{sum}\ \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \textsc{par}\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\textsc{com}\ \frac{P \xrightarrow{u\tilde{x}} P',\ \ Q \xrightarrow{\overline{u}\tilde{y}} Q',\ \ |\tilde{x}| = |\tilde{y}|}{P \mid Q \xrightarrow{\{\tilde{x}=\tilde{y}\}} P' \mid Q'} \qquad \textsc{scope}\ \frac{P \xrightarrow{\varphi} P',\ z\,\varphi\,x,\ z \neq x}{(z)P \xrightarrow{\varphi\backslash z} P'\{x/z\}}$$

$$\textsc{pass}\ \frac{P \xrightarrow{\alpha} P',\ z \notin \mathrm{n}(\alpha)}{(z)P \xrightarrow{\alpha} (z)P'} \qquad \textsc{open}\ \frac{P \xrightarrow{(\tilde{y})a\,\tilde{x}} P',\ z \in \tilde{x} - \tilde{y},\ a \notin \{z, \overline{z}\}}{(z)P \xrightarrow{(z\tilde{y})a\,\tilde{x}} P'}$$

$$\textsc{match}\ \frac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'} \qquad \textsc{mismatch}\ \frac{P \xrightarrow{\alpha} P',\ x \neq y}{[x \neq y]P \xrightarrow{\alpha} P'}$$

Table 1. The Fusion Calculus: Laws of action.

The COM rule results in a fusion action rather than a substitution, and the SCOPE rule entails a substitution of the scoped name z for a nondeterministically chosen name x related to it by φ. For the purpose of the equivalence defined below it will not matter which such x replaces z. The only rule dealing with bound actions is OPEN. Using structural congruence, pulling the relevant scope to the top level, we can still infer e.g. $P \mid (x)ayx\,.\,Q \xrightarrow{(x)ayx} P \mid Q$ using PREF and OPEN (provided $x \notin \mathrm{fn}(P)$, otherwise an alpha-conversion is necessary). It is clear that for the purpose of the semantics, fusion prefixes can be regarded as derived forms since $\{\tilde{y} = \tilde{z}\} . P$ has exactly the same transitions as $(u)(\overline{u}\tilde{y} . \mathbf{0} \mid u\tilde{z} . P)$ when $u \notin \mathrm{fn}(P)$. (In the same way the τ prefix can be regarded as derived in CCS and in the π-calculus.)

For further examples and explanations we refer the reader to [7] and [14].

2.2 Equivalences

From [7] we recall the definition of strong bisimulation in the fusion calculus.

Definition 5. *A substitution σ* agrees with *the fusion φ if $\forall x, y : x \varphi y \Leftrightarrow \sigma(x) = \sigma(y)$. A* substitutive effect *of a fusion φ is a substitution σ agreeing with φ such that $\forall x, y : \sigma(x) = y \Rightarrow x \varphi y$ (i.e., σ sends all members of the equivalence class to one representative of the class). The only substitutive effect of a communication action is the identity substitution.*

Definition 6. *A* bisimulation *is a binary symmetric relation $\mathcal{S}$ between agents such that $P \mathcal{S} Q$ implies: If $P \xrightarrow{\gamma} P'$ with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$ then $Q \xrightarrow{\gamma} Q'$ and $P'\sigma \mathcal{S} Q'\sigma$ for some substitutive effect σ of γ. P is bisimilar to Q, written $P \dot{\sim} Q$, if $P \mathcal{S} Q$ for some bisimulation $\mathcal{S}$.*

This definition differs from *ground* bisimulation only in the treatment of fusion actions. A fusion $\{x = y\}$ represents an obligation to make x and y equal everywhere. Therefore, if γ above is such a fusion, it only makes sense to relate P' and Q' when a substitution $\{y/x\}$ or $\{x/y\}$ has been performed. Note that it does not matter which substitution we choose, since $P\{x/y\} \dot{\sim} Q\{x/y\}$ implies $P\{y/x\} \dot{\sim} Q\{y/x\}$, by the simple fact that $P\{x/y\}\{y/x\} \equiv P\{y/x\}$ and that bisimulation is closed under injective substitutions.

The *weak* simulation, bisimulation and bisimilarity $\dot{\approx}$ are as usual defined by replacing $Q \xrightarrow{\gamma} Q'$ by $Q \overset{\gamma}{\Longrightarrow} Q'$ in Definition 6. For the case that γ is a fusion the definition of $\overset{\gamma}{\Longrightarrow}$ requires some care. The intuition is that $\overset{\varphi}{\Longrightarrow}$ represents a sequence of actions with "observable content" φ. But fusions are "observable" only through their substitutive effects. They have no subject and the environment cannot synchronize with them and keep track of, e.g., how many have been performed. Therefore we allow $\overset{\varphi}{\Longrightarrow}$ to be a sequence of fusions whose aggregated substitutive effect is the same as that of φ.

Definition 7. *Define the* composition *of two transitions, $\circ$, by $P(\xrightarrow{\gamma} \circ \xrightarrow{\gamma'})Q$ iff there exists an agent P' such that $P \xrightarrow{\gamma} P'$ and $P'\sigma_\gamma \xrightarrow{\gamma'} Q$, where σ_γ is a substitutive effect of γ. Define the* conjunction *of two fusions φ and ψ, written $\varphi \wedge \psi$, to be the least equivalence relation containing φ and ψ. Define the* weak transition $\overset{\gamma}{\Longrightarrow}$ *by the following: $P \overset{\gamma}{\Longrightarrow} Q$ means that for some $n \geq 0$, $P \xrightarrow{\gamma_1} \circ \cdots \circ \xrightarrow{\gamma_n} Q$ and either of*

1. *γ is a communication and $\gamma = \gamma_i$ for some i and $\gamma_j = \mathbf{1}$ for all $j \neq i$, or*
2. *γ and all γ_i are fusions and $\gamma = \gamma_1 \wedge \cdots \wedge \gamma_n$. Here we allow $n = 0$ where the empty conjunction is $\mathbf{1}$, in other words $P \overset{\mathbf{1}}{\Longrightarrow} P$ holds for all P.*

An illuminating exercise is to verify that

$$\{x = y\} . [x = y]\{v = w\} . P \quad \dot{\approx} \quad \{x = y\} . \{v = w\} . P \ + \ \{x = y, v = w\} . P$$

but not $\{x = y\} . \{v = w\} . P \mathrel{\dot{\approx}} \{x = y, v = w\} . P$ since the RHS cannot simulate the transition $\xrightarrow{\{x=y\}}$. It is easy to prove that $\dot{\sim}$ and $\dot{\approx}$ are indeed equivalences (transitivity for $\dot{\approx}$ requires some care) and that they are not congruences. To find the congruences we close the bisimulations under arbitary substitutions:

Definition 8. *A* (weak) hyperbisimulation *is a substitution closed (weak) bisimulation, i.e., a (weak) bisimulation $\mathcal{S}$ with the property that $P \mathrel{\mathcal{S}} Q$ implies $P\sigma \mathrel{\mathcal{S}} Q\sigma$ for any substitution σ. Two agents P and Q are* (weakly) hyperequivalent, *written $P \sim Q$ ($P \approx Q$), if they are related by a (weak) hyperbisimulation.*

Theorem 9. [7] *Hyperequivalence is the largest congruence contained in bisimilarity.*

The corresponding relation does not quite hold for weak hyperequivalence. But we here show that it holds if we replace the summation operator by *guarded* summation $\sum \alpha_i . P_i$. (This is as expected, also in CCS and in the π-calculus weak bisimulation equivalence fails to be a congruence for the same reason.)

Theorem 10. *With guarded summation, weak hyperequivalence is the largest congruence in weak bisimilarity.*

This is proved the same way as the strong case in [7]: we construct a context which can perform any relevant substitutions at any time, and show that if two agents are equivalent in that context, which they must be if they are congruent, then they are also weakly hyperequivalent. The context used is the same as in [7]. Due to space limitations, we refer to the first author's PhD thesis [14] for the full proof.

3 The Barbed Congruences

We shall here provide a *reduction* semantics for the fusion calculus and prove that it in a precise way corresponds to the transition semantics. This lends credibility to our transition laws and equivalence definitions. A reduction semantics is also in some respects easier to comprehend, and facilitates a comparison with the ρ-calculus in the next section. We will here use the standard idea of *barbed bisimulation* [5] and prove that the largest congruence coincides with hyperequivalence.

The reduction semantics of the fusion calculus is given by the rules in Table 2, where again structurally equivalent agents are considered the same. We write $\tilde{M}, \tilde{N}$ for a sequence of match and mismatch operators. Here and in the following we use $\mathrm{dom}(\sigma) = \{u : \sigma(u) \neq u\}$ and $\mathrm{ran}(\sigma) = \{\sigma(u) : \sigma(u) \neq u\}$. The reductions of an appropriately scoped fusion prefix are obtained by viewing a fusion prefix $\{\tilde{x} = \tilde{y}\} . P$ as defined by $(u)(\overline{u}\,\tilde{x} \mid u\,\tilde{y} . P)$ where u is fresh.

Proposition 11. *$P \longrightarrow Q$ iff $P \xrightarrow{\mathbf{1}} Q$.*

The implication to the right is a simple induction over the reduction rules; the implication to the left is an induction over transition rules using a slightly more general induction hypothesis.

We now want to construct a congruence relation based on a minimal power of observation, such that it coincides with the labelled bisimulation congruence. If there are *no* observables, the resulting *reduction congruence* is very weak, but turns out

$$(\tilde{z})\Big(R \mid (\cdots + \tilde{M}u\,\tilde{x}\,.\,P) \mid (\tilde{N}\overline{u}\,\tilde{y}\,.\,Q + \cdots)\Big) \longrightarrow (R \mid P \mid Q)\sigma$$
$$\text{if } |\tilde{x}| = |\tilde{y}|, \tilde{M} \Leftrightarrow \tilde{N} \Leftrightarrow \mathsf{true},$$
$$\sigma \text{ agrees with } \{\tilde{x} = \tilde{y}\}, \mathrm{ran}(\sigma) \cap \tilde{z} = \emptyset, \text{ and } \mathrm{dom}(\sigma) = \tilde{z}$$
$$\frac{P \longrightarrow P'}{[x = x]P \longrightarrow P'} \quad \frac{P \longrightarrow P',\ x \neq y}{[x \neq y]P \longrightarrow P'} \quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \quad \frac{P \longrightarrow P'}{P + Q \longrightarrow P'} \quad \frac{P \longrightarrow P'}{(x)P \longrightarrow (x)P'}$$

Table 2. Reduction rules for the fusion calculus.

to be sufficient for divergence-free CCS [5]. For possibly divergent CCS processes, it is sufficient to be able to observe the possibility of communication [5], while in the π-calculus we must also observe the subject of the communication [10]. With the addition of fusion actions, you may suspect that these need to be observable. This is not the case: we can already observe fusions indirectly using the match and mismatch operators. The observation predicate below is thus the same as for the π-calculus.

Definition 12.

$x\tilde{y}\,.\,P \downarrow x$	$[x = x]P \downarrow y$ *if* $P \downarrow y$
$\overline{x}\,\tilde{y}\,.\,P \downarrow x$	$[x \neq z]P \downarrow y$ *if* $P \downarrow y$ *and* $x \neq z$
$(P \mid Q) \downarrow x$ *if* $P \downarrow x$ *or* $Q \downarrow x$	$(x)P \downarrow y$ *if* $P \downarrow y$ *and* $x \neq y$
$(P + Q) \downarrow x$ *if* $P \downarrow x$ *or* $Q \downarrow x$	$A(\tilde{y}) \downarrow z$ *if* $A(\tilde{x}) \stackrel{def}{=} P$, $\lvert\tilde{y}\rvert = \lvert\tilde{x}\rvert$ *and* $P\{\tilde{y}/\tilde{x}\} \downarrow z$

We repeat the standard definition of barbed bisimulation from [10]:

Definition 13. *A* barbed bisimulation *is a symmetric binary relation $\mathcal{S}$ between agents such that $P\ \mathcal{S}\ Q$ implies:*
1. If $P \longrightarrow P'$ then $Q \longrightarrow Q'$ and $P'\ \mathcal{S}\ Q'$.
2. If $P \downarrow x$ for some x, then $Q \downarrow x$.
P is barbed bisimilar to Q, written $P \stackrel{\cdot}{\sim}_b Q$, if $P\ \mathcal{S}\ Q$ for some barbed bisimulation $\mathcal{S}$.

We note that barbed bisimulation is a very weak equivalence: not only is $u\,.\,v\,.\,\mathbf{0} \stackrel{\cdot}{\sim}_b u\,.\,\mathbf{0}$, but in the fusion calculus also $\varphi\,.\,P \stackrel{\cdot}{\sim}_b \mathbf{0}$ for any P and φ different from $\mathbf{1}$, since a non-inert fusion prefix has neither reductions nor observations. By closing the bisimulation under contexts we obtain a more interesting relation:

Definition 14. *Two agents P and Q are* barbed congruent*, written $P \sim_b Q$, if for all contexts $C[\cdot]$, it holds that $C[P] \stackrel{\cdot}{\sim}_b C[Q]$.*

For CCS, the barbed congruence coincides with strong bisimulation [5]. For the π-calculus, it coincides with the early bisimulation congruence [10]. For the fusion calculus, we here show that it coincides with hyperequivalence:

Theorem 15. *$P \sim Q$ iff $P \sim_b Q$.*

Proof. *Every hyperbisimulation is clearly a barbed bisimulation, so $\sim \subseteq \stackrel{\cdot}{\sim}_b$, and $P \sim Q \Rightarrow C[P] \sim C[Q]$ for all contexts $C[\cdot]$, since $\sim$ is a congruence, so $C[P] \stackrel{\cdot}{\sim}_b C[Q]$, which means that $P \sim_b Q$. This means $\sim \subseteq \sim_b$. We know from [7] that $\sim$ is the largest congruence contained in bisimulation, so we need only show that barbed congruence is also contained in bisimulation to complete the proof. This is done in Lemma 16 below.* $\square$

Lemma 16. *Barbed congruence is contained in bisimulation.*

The proof goes by constructing a context $C[\cdot]$ such that $\{(P,Q) : C[P] \sim_b C[Q]\}$ is a bisimulation. The context used is similar to (but simpler than) the one used for the weak case in Lemma 18 below; we omit the details.

To define the weak barbed bisimulation and congruence, we change $Q \longrightarrow Q'$ to $Q \longrightarrow^* Q'$ and $Q \downarrow x$ to $Q \longrightarrow^* \downarrow x$ (written $Q \Downarrow x$) in Definition 13. We write $P \dot{\approx}_b Q$ if P and Q are weak barbed bisimilar, and $P \approx_b Q$ if they are weak barbed congruent, again under the restriction to guarded summation.

Theorem 17. *$P \approx_b Q$ iff $P \approx Q$.*

The proof is just like that for Theorem 15, and needs Theorem 10 of Section 2 and Lemma 18 below.

Lemma 18. *Weak barbed congruence is contained in weak bisimulation.*

Proof. The proof is reminiscent of the corresponding proof for the π-calculus [10], but simpler. We construct a family of contexts $C^{\tilde{z}}_{n,k}[\cdot]$ such that

$$\mathcal{R} = \{(P,Q) : C^{\tilde{z}}_{n,k}[P] \approx_b C^{\tilde{z}}_{n,k}[Q], \mathrm{fn}(P,Q) \subseteq \tilde{z}, n = |\tilde{z}|, k = \mathrm{ma}(P,Q)\}$$

is a weak bisimulation, where $\mathrm{ma}(P,Q)$ *is the maximal arity of communication actions in P and Q.*

In the definition below we postulate for each name u a fresh name u'; for each finite sequence of names $\tilde{y}$ a fresh name y''; and for each finite equivalence relation e on names a fresh name m_e. Furthermore, $\{c, s, in, out, d_n : n \geq 0\}$ contains fresh names, each M_e is a sequence of match and mismatch operators agreeing with the equivalence relation e, and $\mathrm{eq}(\tilde{z})$ stands for the set of equivalence relations on names in $\tilde{z}$. The definition of $Count_n$ implictly uses an infinite number of arguments (remember that all free names must be in the argument list of the identifier). This can be avoided by parameterizing the context on the length r of the argument list and letting $\mathcal{R}$ include contexts for all r.

$$\begin{aligned}
C^{\tilde{z}}_{n,k}[\cdot] &\equiv (\tilde{z})(\cdot \mid V^k_n(\tilde{z}) \mid Count_n) \\
Count_n &\stackrel{def}{=} d_n + c\,.\,Count_{n+1} \\
V^k_n(\tilde{z}) &\stackrel{def}{=} \sum_{u_i \in \tilde{z}, |\tilde{y}| \leq k} (\tilde{y}) u_i\, \tilde{y}\,.\,\overline{c}\,.\,(u'_i + out + y'' + \mathbf{1}\,.\,V^k_{n+1}(\tilde{z}\tilde{y})) \\
&\quad + \sum_{u_i \in \tilde{z}, |\tilde{y}| \leq k} (\tilde{y}) \overline{u_i}\, \tilde{y}\,.\,\overline{c}\,.\,(u'_i + in + y'' + \mathbf{1}\,.\,V^k_{n+1}(\tilde{z}\tilde{y})) \\
&\quad + s \\
&\quad + \sum_{e \in \mathrm{eq}(\tilde{z})} M_e m_e
\end{aligned}$$

Now, for $(P,Q) \in \mathcal{R}$,

1. *If $P \stackrel{(\tilde{x})u\tilde{w}}{\longrightarrow} P'$, we must find a Q' such that $Q \stackrel{(\tilde{x})u\tilde{w}}{\Longrightarrow} Q'$ and $(P',Q') \in \mathcal{R}$. $C^{\tilde{z}}_{n,k}[P] \stackrel{\mathbf{1}}{\longrightarrow} \stackrel{\mathbf{1}}{\longrightarrow} (\tilde{z})(\tilde{x})(P' \mid W \mid Count_{n+1}) \equiv S$ where W is the appropriate derivative of $V^k_n(\tilde{z})$. Now, $S \downarrow d_{n+1}$, so for $C[Q]$ to simulate the reduction, it must reduce involving Q and V, but it cannot let Q communicate more than once. Furthermore, $S \downarrow u'_i$ and $S \downarrow in$, so $Q \stackrel{\alpha}{\Longrightarrow} Q'$ with* $\mathrm{subj}(\alpha) = u_i$*, and $S \downarrow y''$, so* $\mathrm{obj}(\alpha) = \tilde{y} = \tilde{w}$*,*

where $\tilde{x} \subseteq \tilde{w}$ is fresh. Then $C[Q] \overset{(\tilde{x})a\tilde{w}}{\Longrightarrow} (\tilde{z})(\tilde{x})(Q' \mid W \mid Count_{n+1}) \equiv T$. Finally the reduction $S \xrightarrow{\mathbf{1}} C^{\tilde{z}\tilde{x}}_{n+1,k}[P']$ must be simulated by $T \xrightarrow{\mathbf{1}} C^{\tilde{z}\tilde{x}}_{n+1,k}[Q']$ to preserve the observability of s and d_{n+1}.

2. *If $P \xrightarrow{(\tilde{x})\overline{u}\,\tilde{w}} P'$ the argument is symmetric.*
3. *If $P \xrightarrow{\varphi} P'$, then $C^{\tilde{z}}_{n,k}[P] \xrightarrow{\mathbf{1}} C^{\tilde{z}}_{n,k}[P']\sigma \downarrow m_e$ for σ agreeing with φ and m_e encoding the equivalence relation of names in $\tilde{z}$ agreeing with φ. To preserve this observation, it must be that $Q \overset{\varphi}{\Longrightarrow} Q'$, i.e., Q can perform a sequence of fusion actions which "sum up" to the same effect as φ, and then $C^{\tilde{z}}_{n,k}[Q] \overset{\mathbf{1}}{\Longrightarrow} C^{\tilde{z}}_{n,k}[Q']\sigma \downarrow m_e$; if $Q \overset{\psi}{\Longrightarrow} Q'$ for some ψ different from φ, $m_f \neq m_e$ will be observable.*

Thus, the relation $\mathcal{R}$ is a weak bisimulation. □

4 Encoding constraint calculi

The ρ-calculus of Niehren and Müller [6] is a concurrent calculus parameterized by a constraint system, which serves as a foundation for the concurrent constraint language Oz [13]. Its development has been inspired by process calculi such as the π-calculus, but analysis methods such as equivalence checking or model checking have not been applied to it. This is partly because the calculus contains elements such as logical variables with no direct counterparts in other process calculi.

In the ρ-calculus, the constraint store can be seen as transparently distributed over the terms, in contrast with the monolithic constraint store of, e.g., [11]. The constraint store can be asked if a formula or its negation is entailed by using a conditional term, and a constraint is implicitly added when it is not guarded by a conditional or abstraction.

In this section we relate the fusion calculus to three basic instances of the ρ-calculus: $\rho(x = y)$, which uses constraints over name equations and conjunction; $\rho(x = y, \mathcal{C})$, which adds constants to the constraint system; and $\rho(x = y, x \neq y)$, which adds inequalities. We relate the calculi by giving a simple encoding of a term E in one of the instances of the ρ-calculus such that if E can reduce to another term F, then its encoding can reduce to the encoding of F, and vice versa. Correspondence between observations and of convergence properties is also shown.

4.1 The ρ-calculus

We give a brief presentation of the syntax and semantics of the ρ-calculus. For a fuller treatment we refer the reader to [6].

There are three basic entities in the ρ-calculus: *variables*, *constraints* and *terms*. Variables are ranged over by lower-case letters, constraints by $\phi, \varphi, \psi, \ldots$, and terms by $E, F, \ldots$. Their syntax is given by the following BNF equation:

$$E ::= \top \;\Big|\; E_1 \wedge E_2 \;\Big|\; \exists x E \;\Big|\; x\tilde{y} \;\Big|\; x{:}\,\tilde{y}/E \;\Big|\; \texttt{if}\ \varphi\ \texttt{then}\ E_1\ \texttt{else}\ E_2 \;\Big|\; \phi$$

The terminology in the following explanation comes from [6]: Briefly, $\top$ represents an inactive term; each term in a *composition* $E_1 \wedge E_2$ can reduce in parallel; a *declaration* $\exists x E$ introduces a new variable x with scope E; an *abstraction* $x{:}\,\tilde{y}/E$ can communicate with an *application* $x\tilde{z}$, producing the new term $E\{\tilde{z}/\tilde{y}\}$. A *conditional* `if` φ `then` E_1 `else` E_2 tests if the current constraint store entails the condition φ or its negation. A *constraint* term ϕ is added to the current constraint store. We write $\phi \models \psi$ if $\phi \Rightarrow \psi$ is valid in all models of the constraint system. The constraint $\dot{\top}$ stands for logical truth, and conjunction of constraints is written $\dot{\wedge}$. The terms $\exists y E$ and $x{:}\,\tilde{y}/E$ bind y and $\tilde{y}$, respectively; we write $\mathrm{fv}(E)$ for all names in E which are not bound.

The formal semantics of the ρ-calculus is given by a structural congruence in Table 3 and reduction rules in Table 4. Again structurally congruent terms are considered the same, i.e., if $E \equiv E'$ and $F \equiv F'$ and $E \longrightarrow F$ then also $E' \longrightarrow F'$.

$E \wedge \dot{\top} \equiv E$	$E \wedge \top \equiv E$	$E \wedge F \equiv F \wedge E$	$E \wedge (F \wedge G) \equiv (E \wedge F) \wedge G$
$\exists x\, \dot{\top} \equiv \dot{\top}$	$\exists x \top \equiv \top$	$\exists x \exists y E \equiv \exists y \exists x E$	$E \wedge \exists x F \equiv \exists x(E \wedge F)$ if $x \notin \mathrm{fv}(E)$
$\phi \wedge \psi \equiv \phi \,\dot{\wedge}\, \psi$	$\phi \equiv \varphi$ if $\phi \Leftrightarrow \varphi$		

Table 3. Structural congruence relation of the ρ-calculus.

$\phi \wedge x{:}\,\tilde{y}/E \wedge z\tilde{w} \longrightarrow \phi \wedge x{:}\,\tilde{y}/E \wedge E\{\tilde{w}/\tilde{y}\}$	if $\phi \models x = z$ and $\lvert\tilde{y}\rvert = \lvert\tilde{w}\rvert$	$\dfrac{E \longrightarrow F}{\exists x E \longrightarrow \exists x F}$
$\phi \wedge$ `if` ψ `then` E `else` $F \longrightarrow \phi \wedge E$	if $\phi \models \psi$	
$\phi \wedge$ `if` ψ `then` E `else` $F \longrightarrow \phi \wedge F$	if $\phi \models \neg\psi$	$\dfrac{E \longrightarrow E'}{E \wedge F \longrightarrow E' \wedge F}$

Table 4. Reduction rules of the ρ-calculus.

We say that a constraint is *guarded* if it occurs in the body of an abstraction or the clauses of a conditional. We show that a ρ-calculus term E can be written in a head normal form with all unguarded constraints merged together. For example, $x = y \wedge \exists z(z = w \wedge aw)$ has the head normal form $\exists z(x = y \,\dot{\wedge}\, z = w \wedge aw)$.

Definition 19. *A ρ-calculus term E is in* head normal form *if E is of the form $\exists\tilde{x}(\phi \wedge E')$, where E' has no unguarded constraints and $\tilde{x} \subseteq \mathrm{fv}(\phi)$.*

In this head normal form ϕ can be thought of as representing the store.

Proposition 20. *Any ρ-calculus term E is structurally congruent to a term F which is in head normal form.*

The proof is by structural induction on terms.

We further define the observation predicate of the ρ-calculus in a straightforward way:

Definition 21. $x\tilde{y} \downarrow x \qquad x{:}\,\tilde{y}/E \downarrow x$
$(E \mid F) \downarrow x$ *if* $E \downarrow x$ *or* $F \downarrow x$
$\exists x E \downarrow y$ *if* $E \downarrow y$ *and* $x \neq y$

4.2 Constraints over name equations

We first instantiate the constraint system in ρ with simple name equations and conjunction. This calculus is $\rho(x=y)$ of [6]. We give a simple encoding for it into the fusion calculus in Table 5. We here use the standard replication operator definable through recursion by $!P \stackrel{def}{=} P \mid !P$. For simplicity, we only describe the encoding where the condition of a conditional is of the form $x=y$. The general case can also be handled, e.g. by nesting conditionals. The encoding of a ρ-calculus name x is written $\underline{x}$. In Table 5 it suffices to define $\underline{x}=x$, assuming a unique name in the fusion calculus for each name in the ρ-calculus. Note that in $\rho(x=y)$, it is *never* the case that $\phi \models \neg(x=y)$ for any ϕ, so the "else" branch of a conditional can be completely ignored.

$$\begin{array}{rcl}
\underline{x} & \equiv & x \\
[\![\top]\!] & \stackrel{def}{=} & \mathbf{0} \\
[\![x\tilde{w}]\!] & \stackrel{def}{=} & \overline{x}\,\underline{\tilde{w}} \\
[\![x{:}\,\tilde{y}/E]\!] & \stackrel{def}{=} & !\,(\underline{\tilde{y}})x\,\underline{\tilde{y}}\,.\,[\![E]\!] \\
[\![x_1=y_1 \mathbin{\dot\wedge} \cdots \mathbin{\dot\wedge} x_n=y_n]\!] & \stackrel{def}{=} & \{\underline{x_1}=\underline{y_1},\cdots,\underline{x_n}=\underline{y_n}\} \\
[\![E \wedge F]\!] & \stackrel{def}{=} & [\![E]\!] \mid [\![F]\!] \\
[\![\exists x E]\!] & \stackrel{def}{=} & (\underline{x})[\![E]\!] \\
[\![\texttt{if } x=y \texttt{ then } E \texttt{ else } F]\!] & \stackrel{def}{=} & [x=y]\mathbf{1}\,.\,[\![E]\!]
\end{array}$$

Table 5. The encoding of $\rho(x=y)$.

For the correctness of the encoding, we treat top-level constraints specially, and define $\mathcal{V}(\exists\tilde{u}(\phi\wedge E)) \stackrel{def}{=} (\underline{\tilde{u}})[\![E]\!]\sigma_\phi$ where $\exists\tilde{u}(\phi\wedge E)$ is in head normal form, σ_ϕ agrees with $[\![\phi]\!]$ and $\mathrm{ran}(\sigma_\phi)\cap\underline{\tilde{u}}=\emptyset$. In other words, the effect of the equations of the store is applied onto E. For example, $\mathcal{V}(\exists x(x=y\wedge axy))$ is $\overline{a}\,yy$.

The correctness is stated by the following theorem. The addition of scopes $\tilde{z}$ is a technical device, necessary to turn guarded fusions of free names in $\mathcal{V}(E)$ into reductions.

Theorem 22. *For E,F terms of $\rho(x=y)$ in head normal form,*

1. *If $E \longrightarrow F$, then for some $\tilde{z}$, $(\tilde{z})\mathcal{V}(E) \longrightarrow^{+} P \approx \mathcal{V}(F)$.*
2. *For any $\tilde{z}$, if $(\tilde{z})\mathcal{V}(E) \longrightarrow^{*} P$, then $E \longrightarrow^{*} F$ and $P \approx (\tilde{z})\mathcal{V}(F)$.*
3. *$E\downarrow x$ iff $\mathcal{V}(E)\downarrow x'$, where $E=\phi\wedge E'$ and $\phi\models x=x'$.*
4. *If E converges to F, then for some $\tilde{z}$, $(\tilde{z})\mathcal{V}(E)$ converges to $P\approx\mathcal{V}(F)$.*
5. *For any $\tilde{z}$, if $(\tilde{z})\mathcal{V}(E)$ converges to P and P has no fusion transitions, then E converges to F with $P\approx(\tilde{z})\mathcal{V}(F)$.*

The proof is by inductions on the depths of inferences.

4.3 Adding constants

Adding constants to the constraint system of $\rho(x=y)$ gives a system which is closer to the γ-calculus of Smolka [12]. Here, the variables are divided into two families: normal variables and constants, and the axiom $\forall x, y \in \mathcal{C}, x \not\equiv y : x \neq y$ is added to the constraint system, where $\mathcal{C}$ is the family of constants. This means that two constants which are not literally the same are always different. This instance of the ρ-calculus is called $\rho(x = y, \mathcal{C})$. Below we encode it into the fusion calculus. Here and in the following encoding, we write $[x = y](P, Q)$ for the fusion calculus agent $[x = y]P + [x \neq y]Q$.

As we have seen in the encoding of $\rho(x = y)$, variables of the ρ-calculus correspond directly to names in the fusion calculus. Constants, however, are explicitly encoded in the fusion calculus by adding a *handler process* for each constant, which can tell its environment that it is a constant. The encoding of a conditional can interrogate these handler processes to find out if $\neg\varphi$ is entailed. (Cf. [15], where the γ-calculus was encoded in the π-calculus using much more complex handler processes for constants, variables and updated variables.)

For each variable or constant u in $\rho(x = y, \mathcal{C})$ we introduce *two* names in the encoding: u and u_c. We redefine $\underline{u}$ to refer to these two together. u_c is used for communicating with a handler process for u, if it exists. A handler process for u can synchronize repeatedly on u_c, thereby signalling that u is a constant. We update the encoding in Table 5 by Table 6.

$$
\begin{aligned}
\underline{u} &\equiv u, u_c \\
C(a_c) &\stackrel{def}{=} !\,\overline{a_c} \\
[\![\exists a E]\!] &\stackrel{def}{=} (\underline{a})(C(a_c) \mid [\![E]\!]) \text{ if } a \in \mathcal{C} \\
[\![\exists x E]\!] &\stackrel{def}{=} (\underline{x})[\![E]\!] \qquad\qquad \text{if } x \notin \mathcal{C} \\
[\![\texttt{if } x = y \texttt{ then } E \texttt{ else } F]\!] &\stackrel{def}{=} [x = y](\mathbf{1} \,.\, [\![E]\!], x_c \,.\, y_c \,.\, [x = y](\mathbf{1} \,.\, [\![E]\!], \mathbf{1} \,.\, [\![F]\!]))
\end{aligned}
$$

Table 6. The encoding of $\rho(x = y, \mathcal{C})$.

The encoding of a conditional deserves a short explanation: If x and y are the same, we choose the E branch directly. Otherwise, to check if $\neg(x = y)$ is entailed, we must verify that both x and y are constants. Then we must *again* test if $x = y$ or not, for although they were not equal, a concurrent constraint may since have fused them.

For the correctness of this encoding, we must put the encoded terms in a context where all free a_c are scoped, and all free constants have a handler process. When I is a finite set we use $\prod_{i \in I} P_i$ to mean the Composition of all P_i for $i \in I$, and define $\mathcal{V}(\exists\tilde{u}(\phi \wedge E)) \stackrel{def}{=} (\underline{\tilde{u}})(\tilde{v}_c)((\prod_{a \in \tilde{a}} C(a_c) \mid [\![E]\!])\sigma_\phi)$ where $\exists\tilde{u}(\phi \wedge E)$ is in head normal form, $\tilde{v} = \mathrm{fv}(\exists\tilde{u}(\phi \wedge E))$, $\tilde{a} = \tilde{v} \cap \mathcal{C}$, σ_ϕ agrees with $[\![\phi]\!]$ and $\mathrm{ran}(\sigma_\phi) \cap \underline{\tilde{u}} = \emptyset$. Theorem 22 holds for $\rho(x = y, \mathcal{C})$ with this definition of $\mathcal{V}$.

An inconsistent constraint store can be detected by redefining $C(a_c) \stackrel{def}{=} \overline{a_c} \,.\, C(a_c) + a_c \,.\, \mathit{fail}$. Then *fail* is a weak observation if the store gets inconsistent. Also, the *cell* construct of [12, 6] can be encoded by $\mathit{Cell}(x, u) \stackrel{def}{=} (w)x\,uw \,.\, \mathit{Cell}(x, w)$, nicely utilizing the free input of the fusion calculus.

4.4 Name inequations

Instead of constants, we here add explicit inequations to the constraint system, acquiring $\rho(x = y, x \neq y)$. Although $x \neq y$ could be encoded in $\rho(x = y, \mathcal{C})$ as $\exists a \exists b(x =$

$a \wedge y = b$), in $\rho(x = y, x \neq y)$ we avoid splitting the variables into constants and normal variables. In this encoding we also need additional names and processes, but the processes now explicitly encode inequations rather than constants.

$$
\begin{aligned}
\underline{x} &\equiv x, x_d \\
D(\underline{x}, \underline{y}) &\stackrel{def}{=} !\,(w,n)x_d\, wn\,.\,[w = y](\overline{n}, \overline{x_d}\, wn) \\
[\![\texttt{if } x = y \texttt{ then } E \texttt{ else } F]\!] &\stackrel{def}{=} (n)(\overline{x_d}\, yn \mid \overline{y_d}\, xn \mid ([x = y](\mathbf{1}\,.\,[\![E]\!], n\,.\,[\![F]\!]))) \\
[\![x_1 \neq y_1 \,\dot{\wedge} \cdots \dot{\wedge}\, x_n \neq y_n]\!] &\stackrel{def}{=} D(\underline{x_1}, \underline{y_1}) \mid \cdots \mid D(\underline{x_n}, \underline{y_n})
\end{aligned}
$$

Table 7. The encoding of $\rho(x = y, x \neq y)$.

The encoding in Table 7 updates the one in Table 5; note again the redefinition of $\underline{x}$. The presence of a process $D(\underline{x}, \underline{y})$ means that the inequation $x \neq y$ is present in the store. This process handles queries along x_d with objects w, n, meaning "is x different from w"? If $w = y$ it replies affirmatively by synchronizing on the supplied name n; otherwise it re-iterates the query to see if some other handler can decide it. This introduces divergence in the encoding, since the new query can be handled by the same process. We regard this as processing internal to the constraint system, which should not be a concern of the external observer. The encoding of a conditional simply asks both names in the equation if they are different.

The correctness again uses a context at top level, which scopes all free x_d. We redefine $\mathcal{V}(\exists\tilde{u}(\phi \,\dot{\wedge}\, \psi \wedge E)) \stackrel{def}{=} (\underline{\tilde{u}})(\tilde{v}_d)[\![\psi \wedge E]\!]\sigma_\phi$ where $\exists\tilde{u}(\phi \,\dot{\wedge}\, \psi \wedge E)$ is in head normal form, ϕ contains only positive constraints and ψ only negative, $\tilde{v} = \mathrm{fv}(\exists\tilde{u}(\phi \,\dot{\wedge}\, \psi \wedge E))$, σ_ϕ agrees with $[\![\phi]\!]$ and $\mathrm{ran}(\sigma_\phi) \cap \underline{\tilde{u}} = \emptyset$. Theorem 22(1-3,5) holds for $\rho(x = y, x \neq y)$ with this definition of $\mathcal{V}$; case 4 holds if we disregard divergences introduced by D agents. The stronger form of case 5 where such divergences are ignored also holds.

By using $D'(\underline{x}, \underline{y}) \stackrel{def}{=} [x = y]\mathit{fail} \mid D(\underline{x}, \underline{y})$ in place of D, we can detect inconsistencies of the store.

5 Conclusion

We have explored the weak and barbed equivalences of the fusion calculus and used them to verify a model of the ρ-calculus. An interesting implication is that the ρ-calculus thereby may gain access to theories and tools developed for process calculi.

There are several lines of further work worth exploring. The correspondence between a ρ-calculus term and its encoding (Theorem 22) can probably be strengthened to imply that two terms are equivalent iff their encodings are equivalent. The question is which equivalence would be relevant in such a result. Convergence is of prime importance in the ρ-calculus and our equivalences do not currently take that into account. Refining the equivalences in that respect is probably necessary.

The fusion calculus as presented here is unsorted, while the ρ-calculus comes in a version where "constants" and "variables" are distinguished. An interesting question is if the sorting systems of the π-calculus can be adapted to accommodate this.

The question whether more complex constraint systems can be handled in the same way is largely open. Clearly the fusion calculus is geared particularly towards constraints on the identity of names. But it may be that unification can be represented cleanly, implying that ordinary logic programming has a good encoding.

Finally, simulation and verification tools for the π-calculus are currently being adapted for the fusion calculus. Using our encodings it is a straightforward task to derive tools also for the ρ-calculus.

References

[1] Y. Fu. A proof-theoretical approach to communication. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *LNCS*, pages 325–335. Springer, 1997.

[2] B. Li. A π-calculus specification of Prolog. In D. Sannella, editor, *Proceedings of ESOP '94*, volume 788 of *LNCS*, pages 379–393. Springer, 1994.

[3] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[4] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.

[5] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[6] J. Niehren and M. Müller. Constraints for free in concurrent computation. In K. Kanchanasut and J.-J. Lévy, editors, *Asian Computer Science Conference*, volume 1023 of *LNCS*, pages 171–186. Springer, 1995.

[7] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS'98*. IEEE, Computer Society Press, June 1998. URL: `http://www.docs.uu.se/~victor/tr/fusion.html`.

[8] J. Parrow and B. Victor. The update calculus. In M. Johnson, editor, *Proceedings of AMAST'97*, volume 1349 of *LNCS*, pages 409–423. Springer, Dec. 1997.

[9] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1997. To appear.

[10] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, LFCS, University of Edinburgh, 1993.

[11] V. A. Saraswat, M. Rinard and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of POPL '91*, pages 333–352. ACM, 1991.

[12] G. Smolka. A foundation for higher-order concurrent constraint programming. In J.-P. Jouannaud, editor, *Constraints in Computational Logics*, volume 845 of *LNCS*, pages 50–72. Springer, Sept. 1994.

[13] G. Smolka. The Oz programming model. In J. van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer, 1995.

[14] B. Victor. *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes.* PhD thesis, Dept. of Computer Systems, Uppsala University, Sweden, June 1998. URL: `http://www.docs.uu.se/~victor/thesis.shtml`.

[15] B. Victor and J. Parrow. Constraints as processes. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 389–405. Springer, 1996.

[16] D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.