CSP is expressive enough for π

A.W. Roscoe

Oxford University Computing Laboratory
{Bill.Roscoe@comlab.ox.ac.uk}

Abstract. Recent results show that Hoare's CSP, augmented by one additional operator, can express every operator whose operational semantics are expressible in a new notation and are therefore "CSP-like". In this paper we show that π-calculus fits into this framework and therefore has CSP semantics. Rather than rely on the machinery of the earlier result we develop a much simpler version from scratch that avoids the extra operator and is sufficient for π-calculus: a much generalised *relabelling* operator that is expressed in terms of the others. We present a number of different options for the semantics of fresh variables, showing how they give semantics that are largely congruent to each other. Finally, we begin the investigation of how these new semantics might be analysed and exploited.

1 Introduction

This paper is dedicated to Sir Tony Hoare on the occasion of his 75th birthday.

When I contributed [15] to the volume celebrating Tony's 60th birthday, CSP was just half as old as it is now. I wrote then how remarkable it was that it should have stood the test of so many challenges unimagined by Tony when he created it, and of how frustrating it was to work academically on something that was so "right first time", since one was denied the usual joys of refining and changing it. I have continued to be surprised by its ability to capture new concepts in modelling concurrency and interaction. The present paper recalls recent work quantifying this expressive power and shows that CSP can both represent the π-calculus and provide the vehicle for a wide range of new semantics for that notation.

I can report, however, that my previous frustration has been reduced, since I have finally found an operator to be missing from Tony's original CSP: this will be described below.

When I first described my new expressibility results to Tony, part of his response was *"Which operators of CCS satisfy your definition? Which ones don't? What about π-calculus?"* This paper answers these questions.

While other languages for concurrent systems are often defined in terms of their operational semantics, the CSP approach [6, 16] has always been to regard behavioural models such as *traces* $\mathcal{T}$, *stable failures* $\mathcal{F}$, *failures-divergences* $\mathcal{N}$ and *infinite traces-failures-divergences* $\mathcal{U}$ as equally important means of expression. Thus any operator must make sense over these *behavioural* models in which details of individual linear runs of the processes are recorded by an observer who cannot, of course, see the internal action τ.

Nevertheless CSP has a well-established operational semantics, and congruence with that is perhaps the main criterion for the acceptability of any new model that is proposed.

Operational semantic definitions of languages have the advantage that they are direct, understandable, and of themselves carry no particular obligation to prove congruence results such as those alluded to above. On the other hand definitions in abstract models, intended to capture the extensional meaning of a program in some sense, have the advantage of "cleanliness" and allow us to reason about programs in the more abstract models. The most immediate benefit of CSP models in this respect is that they bring a theory of refinement, which in turn gives refinement checking (with low complexity at the implementation end, as in FDR) as a natural vehicle for specification and verification.

The author has recently [20] defined what it means for an operational semantics to be *CSP-like*, in the sense that it does not require any basic powers beyond what the semantics of the various CSP operators already have. In this paper we will show that the π-calculus (the version presented in [23]) is CSP-like.

The main result of [20] is that every CSP-like operator can be simulated up to strong bisimulation in CSP extended by one more operator. The proof there constructs a very general but complex "machine" for simulating any such operator. In the present paper we will give significantly more straightforward CSP representations (not needing the additional operator) of the constructs we need to give π-calculus a semantics.

In the next section we recall the definitions of a CSP-like operational semantics and the new operator Θ_A needed to complete the general simulation result, as well as summarising the techniques used in that proof. In the following section we will, since it is needed for π-calculus, show how the usual CSP renaming operator can be extended into a much generalised *relabelling* operator that can nevertheless be expressed in terms of standard CSP operators.

There are three significant issues that arise when attempting to give a CSP semantics to π-calculus. These are choices such as $\tau.P + x(y).Q$ that are resolved by τ, the fact that the CCS parallel operator used in π-calculus is very different to that in CSP, and the requirement that names in π-calculus are generated freshly, without collisions. Since the first two of these arise in CCS, we examine the problem of translating CCS into CSP in Section 4. The third problem is handled using generalised relabelling in Section 5, where the translation into CSP of π-calculus is presented.

There seems to be quite a bit of choice in how one handles freshness in the CSP model, and we present a number of options that give (at least for the best-known CSP models) the same equivalence between π-calculus terms.

Throughout this paper, when talking primarily about CSP, Σ_0 will denote the alphabet that our underlying language of processes uses to communicate with each other and with the external environment. We will frequently need to extend this alphabet to allow us to build CSP models of operators not directly contained in CSP. This extended alphabet will be termed Σ. We will later define a corresponding alphabet Σ_π for the embedding of π-calculus into CSP.

Our main references for CSP and π-calculus are respectively [16] and [23]. Our notation is drawn largely from these.

Acknowledgements

Peter Welch [26] originally persuaded me to look at mobility in CSP. The fact that I managed to create a mobile parallel operator for him was one of the main spurs towards looking for a general expressivity result. My initial work on CCS was inspired by the title of [5]. I had useful conversations with many people about this work, notably Tony Hoare, Michael Goldsmith, Samson Abramsky, Rob van Glabbeek, Gavin Lowe and Michael Mislove. This work has benefited from funding by EPSRC and US ONR.

2 CSP is very expressive

Though originally given semantics in behavioural models such as traces $\mathcal{T}$ and failures-divergences $\mathcal{N}$, CSP has long had a congruent operational semantics [2,4]. By *congruent* here, we mean that the sets of behaviours obtained by observing the LTS created by a process's operational semantics are the same as those calculated in the corresponding behavioural model by a denotational semantics. The operational semantics of CSP and some congruence proofs can be found in [16].

There is a lengthy discussion of what makes an operationally-defined operator *CSP-like* in [20]. The first part of the conclusion is that a CSP-like operator has a two-part *arity* (m, I), where m is the finite number of process arguments that are turned **on** at the start of execution, and I indexes a possibly infinite family of arguments that are initially **off**. (Infinite nondeterministic choice and $?x : A \to P(x)$ for infinite A both have I infinite and $m = 0$. The first action of either of these constructs selects a single one of these **off** operands to turn on, and the rest are discarded.)

When defining a family of operators $\{OP_\lambda \mid \lambda \in \Lambda\}$, the actions of $OP_\lambda(P_1, \ldots, P_{m(\lambda)}, \mathbf{Q})$ are determined by λ and the initial actions of the **on** arguments P_λ: they are all the actions deducible under a set of rules determined by λ. There are two sorts of rule:

- A rule *promoting* a τ action for each **on** argument: these take the form:

$$\frac{P_i \xrightarrow{\tau} P_i'}{OP(P_1, \ldots, P_i, \ldots, P_m, \mathbf{Q}) \xrightarrow{\tau} OP(P_1, \ldots, P_i', \ldots, P_m, \mathbf{Q})}$$

  First suggested for CSP in [13], these are termed *patience rules* by van Glabbeek [25] when giving a set of operational rules that respect weak bisimulation.
- An arbitrary collection of rules based on the visible actions of the P_i. Each such rule is represented as a tuple $(\phi, x, \beta, f, \psi, \chi)$ where
  - ϕ is a partial function from $\{1, \ldots, m(\alpha)\}$ to Σ_0 (the alphabet of the underlying processes). Its meaning is that, in order for this transition to fire, each argument P_j such that $j \in \mathit{dom}(\phi)$ must be able to perform the action $\phi(j)$ and become some P_j'. Note that this imposes no condition if $\mathit{dom}(\phi)$ is empty: this corresponds to an action that the operator can perform without reference to an *on* argument, like the initial a in $a \to P$.
  - x is the action in $\Sigma_0 \cup \{\tau\}$ that $OP_\alpha(\mathbf{P}, \mathbf{Q})$ performs as a consequence of the condition expressed in ϕ being satisfied.
  - β is the index of the operator that forms the result state of this action.
  - f is a total function from $\{1, \ldots, k\}$ for some $k = k(\alpha) \geq 0$ to $I(\alpha)$ that represents, in some chosen order, the indexes of the components of $\mathbf{Q}$ (the **off** arguments), that are started up when the rule fires.
  - $\psi : \{1, \ldots, m(\beta)\} \to \{1, \ldots, m(\alpha) + k(\alpha)\}$ is the (total) function that selects each of the resulting state's **on** arguments. It must be include the whole of $\{m(\alpha) + 1, \ldots, m(\alpha) + k(\alpha)\}$ in its range.
  - $\chi : I(\beta) \to I(\alpha)$ is the total function that selects the **off** arguments of OP_β.

These rules give us all the information we need to form the state that results after the action it generates once we state the following. Whenever an **on** argument P_i is present in the result state, then it is in its original state if $i \notin \mathit{dom}(\phi)$ and, if $\phi(i) = a$ then P_i is in state P_i' such that $P_i \xrightarrow{a} P_i'$ in the result.

To illustrate this way of representing operators, we will show how some CSP+ operators fit into this framework. None of them, in fact, need to be defined together with any other operator apart from the identity **id** whose arity is $(1, \emptyset)$ and which has the rules $\{(\{(1,a)\}, a, \mathbf{id}, \emptyset, \{(1,1)\}, \emptyset) \mid a \in \Sigma_0\}$. Here and below we represent the some functions and partial functions as sets of pairs.

- $a \rightarrow \cdot$ has arity $(0, \{-1\})$ and the single rule $(\emptyset, a, \mathbf{id}, \{(1,-1)\}, \{(1,1)\}, \emptyset)$. We have used a negative number to index the **off** argument since it is convenient way of making sure they are disjoint from the **on** indices. **id** is (the index of) the identity operator. Thus here $k = 1$ (the number of **off** arguments turned **on**) and the resulting operator **id** has no **off** arguments.
- $\Box$ has arity $(2, \emptyset)$ and rules $(\{(1,a)\}, a, \mathbf{id}, \emptyset, \{(1,1)\}, \emptyset)$ and $(\{(2,a)\}, a, \mathbf{id}, \emptyset, \{(1,2)\}, \emptyset)$ for each $a \in \Sigma_0$.
- $\backslash\ X$ has arity $(1, \emptyset)$ and the rules $(\{(1,a)\}, \tau, \backslash\ X, \emptyset, \{(1,1)\}, \emptyset)$ for all $a \in X$ and $(\{(1,a)\}, a, \backslash\ X, \emptyset, \{(1,1)\}, \emptyset)$ for all $a \in \Sigma_0 - X$.
- $\underset{X}{\|}$ has arity $(2, \emptyset)$ and rules $(\{(1,a),(2,a)\}, a, \underset{X}{\|}, \emptyset, \{(1,1),(2,2)\}, \emptyset)$ for all $a \in X$ and both $(\{(1,a)\}, a, \underset{X}{\|}, \emptyset, \{(1,1),(2,2)\}, \emptyset)$ and $(\{(2,a)\}, a, \underset{X}{\|}, \emptyset, \{(1,1),(2,2)\}, \emptyset)$ for all $a \notin X$.
- $\triangle$ has arity $(2, \emptyset)$ and, for each $a \in \Sigma_0$, the rules $(\{(1,a), a, \triangle, \emptyset, \{(1,1),(2,2)\}, \emptyset)$ and $(\{(2,a), a, \mathbf{id}, \emptyset, \{(1,2)\}, \emptyset)$. This is the interrupt operator.

The reader might like to compare these with the conventional (SOS) descriptions of the operational semantics of CSP given in Chapter 7 of [16]. Both express exactly the same semantics when combined with the principle of promoting τs from **on** arguments. In fact, the CSP-like operators are *precisely* those that can be presented in this way. It is, however, possible to describe many operators in the SOS style which cannot be translated in to the above form: those that are not CSP-like.

Consider the following CSP-like operator: Θ_A, which we will read P *throw* Q. It has arity $(1, \{-1\})$ and the rule $(\{(1,a)\}, a, \mathbf{id}, \{(1,-1)\}, \{(1,1)\}, \emptyset)$ for each $a \in A$ as well as $(\{(1,b)\}, b, \Theta_A, \emptyset, \{(1,1)\}, \{(-1,-1)\})$ for each $b \in \Sigma_0 - A$.

$P\ \Theta_A\ Q$ runs the process P until it communicates an event in A – which you might think of as a set of exceptions – at which point it hands control over directly to Q. It has much in common with the interrupt operator $\triangle$, except that here it is an event of P rather than one of Q that triggers the hand-over.

In [19], the author showed that Θ_A adds strictly to the expressive power of the CSP language and that $\triangle$ can be expressed in terms of Θ_A and the rest of CSP.

In [20], the author showed that CSP+ (CSP augmented by Θ_A) is capable of simulating any operator with CSP-like operational semantics: for any such operator $OP(P_1, \ldots, P_m, \mathbf{Q})$ we can define a CSP+ context $C_{OP}(P_1, \ldots, P_m, \mathbf{Q})$ that is strongly bisimilar to it. This is done by building a complex "machine" that can interpret any rule of the form outlined above appropriately, and always have the right set of argument processes turned **on** so that the right τs are promoted by CSP.

One of the most important consequences of this result is the following: *every language whose operators are all CSP-like has a denotational semantics in every denotational model of CSP.* Thus, by showing that a language is CSP-like in this way one simultaneously equips it with many different new semantics.

The established denotational models of CSP take the form of recording one or more sorts of behaviour that an observer might see on a single run of the process: these are *linear* as opposed to *branching* behaviours. In [18, 17], the author defined what a behavioural model of CSP is in the cases of *finite observation* models and *divergence-strict* models. These are congruences that are relational images of two specific models. In the finite observation case this is the model $\mathcal{FL}$ that observes sequences of the form

$$\langle A_0, b_1, A_1, b_2, \ldots, A_{n-1}, b_n, A_n \rangle$$

where the b_i are the visible events performed by the process, and A_i is either the *stable acceptance set* offered by the process in the state from which b_{i+1} occurs, or $\bullet$ meaning that stability was not observed. The final acceptance set can be $\emptyset$, meaning that the process is deadlocked.

In the strict divergence case two further components are added to get a model $\mathcal{FL}^{\Downarrow\omega}$: infinite sequences of the same sort, and finite sequences with the final A_n replaced by $\Uparrow$, meaning that the process divergences (performs an infinite sequence of τ actions). By *strict divergence* we mean that all extensions of any divergence are also considered to be behaviours of the process: no attempt is made to distinguish two processes on the basis of what they can or cannot do after their first possibility to diverge.

The infinite sequence case is necessary to get a congruence for CSP if *unboundedly nondeterministic* constructs are used. We will find that π-calculus is an exception to this rule.

In the original draft of [20], the π-calculus was used as an example to show how general the concept of a CSP-like operational semantics is. We there demonstrated the existence of a CSP semantics for it as a consequence of the above result. The complexity of our simulation machine means, however, that the translation into CSP+ is scarcely clear, and the fact that Θ_A is not actually required becomes obscured.

The present paper therefore refines these techniques so that the translation into standard CSP, and hence the structure of the resulting semantics in CSP's models, become significantly clearer.

3 Generalised relabelling

The main challenge we will have to meet in giving a CSP semantics to π-calculus is dealing with the concept of fresh names. We will find ourselves needing to change names on-the-fly as a process progresses, in a way that is much more flexible than the usual CSP renaming operator $P[\![R]\!]$. Therefore, in this section, we introduce a *generalised relabelling* operator $P\langle\!\langle G\rangle\!\rangle$ and show that it can be expressed using a combination of standard CSP operators.

CSP has two operators that work by changing the labels on a process's actions: hiding and renaming. The first changes a selection of labels to τ and the second maps each action of a process P to a selection from one or more. In each case the mapping on labels does not change as the process progresses, and no action is blocked from occurring.

We can regard both these operators as instances of *relabelling*: replacing each *visible* action of P with an action (or perhaps a choice of actions). We call this "relabelling" rather than "renaming" because x may be τ and so invisible to the environment[1]. As hiding shows, there is no reason why a visible action should not be replaced by τ. It would, however, not be CSP-like even to let the operator notice τ's performed by P: these must always be promoted. In *generalised* relabelling, we will allow two features not seen in either renaming or hiding:

[1] The uses of relabelling we make later only map visible actions to other visible actions.

- We will allow the replacement mapping to vary as the process progresses.
- We will allow the replacement mapping to forbid certain visible actions by P: such actions will map to empty choices of options. So for example $P \underset{A}{\|} STOP$ equivalent to the generalised relabelling that maps every event not in A to itself, and has no image for events in A.

The second of these points is clear cut. The first leaves it open as to what can influence the variation of the mapping. It might be the sequence of visible events that P has performed; it might be the sequence of events that these have been mapped to; or it might be nondeterministic. Or, of course, it might be any combination of these. We will initially consider the first of these, which covers the case of the first generalised renaming HDT used in this paper.

Suppose G is a relation on Σ_0, Σ_0^* and Σ_0^τ, where $(a,t,x) \in G$ says that whenever process P can perform the event a after trace t, the relabelled process $P\langle\!\langle G\rangle\!\rangle$ can perform x. We can give this operator a natural, CSP-like, operational semantics in SOS style as follows:

$$\frac{P \xrightarrow{\tau} P'}{P\langle\!\langle G\rangle\!\rangle \xrightarrow{\tau} P'\langle\!\langle G\rangle\!\rangle} \qquad \frac{P \xrightarrow{a} P', (a,\langle\rangle,x) \in G}{P\langle\!\langle G\rangle\!\rangle \xrightarrow{x} P'\langle\!\langle G/\langle a\rangle\rangle\!\rangle}$$

where $G/t = \{(a,s,x) \mid (s, t\hat{}s, x) \in G\}$.

In our new style of presenting operational semantics, this translates to $\langle\!\langle G\rangle\!\rangle$ having the rule $(\{(1,a)\}, x, \langle\!\langle G/\langle a\rangle\rangle\!\rangle, \emptyset, \{(1,1)\}, \emptyset)$ for each $(a,\langle\rangle,x) \in G$.

Adapting the techniques developed for the most straightforward case of the main theorem of [20], we can re-cast the above operator using two conventional renamings (one one-to-many and one many-to-one), parallel composition with a regulator process, and the hiding of a single event.

We extend the alphabet Σ to include all pairs of the form (a,x) for $a \in \Sigma_0$ and $x \in \Sigma_0 \cup \{\tau\} = \Sigma_0^\tau$ as well as the alphabet Σ_0 of the original processes and the special event tau.

We can define two renamings:

$$E = \{(a,(a,x)) \mid a \in \Sigma_0 \wedge x \in \Sigma_0^\tau\} \qquad C = \{((a,x),\underline{x}) \mid a \in \Sigma_0 \wedge x \in \Sigma_0^\tau\}$$

where $\underline{a} = a$ for $a \in \Sigma_0$, and $\underline{\tau} = tau$.

Clearly $P[\![E]\!][\![C]\!]$ maps every event of P to all events in $\Sigma_0 \cup \{tau\}$, but we can be a lot more selective by running $P[\![E]\!]$ in parallel with a regulator process. Define

$$Reg(G) = \Box\{(a,x) \to Reg(G/\langle a\rangle) \mid (a,\langle\rangle,x) \in G\}$$

It should be clear that

$$(P[\![E]\!] \underset{\Sigma}{\|} Reg(G))[\![C]\!] \setminus \{tau\} \qquad (*)$$

has precisely the same actions as $P\langle\!\langle G\rangle\!\rangle$, and that these two processes are strongly bisimilar on the assumption that the guarded recursion defining $Reg(G)$ does not introduce any τ actions (as discussed in [20], there are two alternative operational semantics for recursion in CSP, one of which introduces a τ for each unfolding and one of which does not).

It is obviously important for practical purposes whether G is, or is not, finitary (respectively finitary relative to P) in the sense that G/s has only finitely many values as s varies (or varies over the traces of P). For we will be able to simulate $Q\langle\!\langle G \rangle\!\rangle$ for general Q (or $P\langle\!\langle G \rangle\!\rangle$ for specific P) using a finite-state regulator just when these apply.

This implementation illustrates how the concept of one-to-many renaming in CSP, as introduced by Hoare, is enormously powerful in allowing us to express a wide variety of constructs that seem at first sight to be beyond what CSP can express. Even more elaborate one-to-many renamings are used in creating the machine in [20].

As well as conventional renaming and hiding (both history-independent in the sense that the relabelling does not depend on what actions have occurred before), the following operations on processes are instances of finitary relabellings:

- Hide every second visible event performed by P.
- Hide all visible events equal to the preceding one.
- Rename all odd-numbered `tock` events to `tick`.
- Prevent all a actions not immediately preceded by member of A.

Rather than deriving the regulator process from G, we can gain maximum freedom in allowing the mapping to vary by instead allowing *Reg* to be any divergence-free process whose alphabet is $\Sigma_0 \times \Sigma_0^\tau$. The relabelling will be said to be *deterministic* just when *Reg* is.

We will use this format for presenting most of the relabellings in this paper, even when they could have been given in terms of relations.

4 CCS

Since π-calculus is built on top of CCS [9, 10], it is useful to consider that language before proceeding to our ultimate goal.

There is a CCS operator, namely $+$, that stands out as *not* being CSP-like, since this can be resolved by a τ action performed by either of its operands. Apart from that, the constant *Nil* is equivalent to the CSP *STOP*; the operational semantics of $\alpha.P$ for $\alpha \neq \tau$ are identical to those of CSP prefix or prefix-choice. The semantics of recursion in CCS is essentially[2] identical to the non-τ version in CSP, and the CCS relabelling operation is a case of CSP renaming. Let us consider the rest of the language: parallel composition $|$, and restriction $\setminus \alpha$.

The structure of Σ_0 (as we again call the set of visible action names used in creating processes) with an operator $\overline{\alpha}$ (with $\overline{\overline{\alpha}} = \alpha$ and $\overline{\alpha} \neq \alpha$) causes no difficulties CSP, although naturally CSP does not understand the relationship between α and $\overline{\alpha}$ unless it is programmed into CSP (as it is in the CSP model of $|$ below).

The CCS restriction operator has semantics

$$\frac{P \xrightarrow{x} P'}{P \setminus \alpha \xrightarrow{x} P' \setminus \alpha}(x \notin \{\alpha, \overline{\alpha}\})$$

Since α is not τ, this is trivially CSP-like, and indeed is equivalent to the CSP construct $P \underset{\{\alpha,\overline{\alpha}\}}{\|} STOP$ as well as being a generalised relabelling of the sort seen in the last section.

[2] The only difference is that CCS allows this definition to be used unconditionally, even on under-defined terms such as $\mu\, p.p$. The translations from CCS to CSP in this section are therefore restricted to the case where all recursions add at least one initial action.

The CCS parallel operator is much more interesting. It has semantics

$$\frac{P \xrightarrow{x} P'}{P \mid Q \xrightarrow{x} P' \mid Q} \qquad \frac{Q \xrightarrow{x} Q'}{P \mid Q \xrightarrow{x} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P' \wedge Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

This is CSP-like, with arity $(2, \emptyset)$ and one set of transition rules corresponding to each of these three clauses: the first two have $(\{(1, \alpha)\}, \alpha, |, \emptyset, \{(1, 1), (2, 2)\}, \emptyset)$ and $(\{(2, \alpha)\}, \alpha, |, \emptyset, \{(1, 1), (2, 2)\}, \emptyset)$ for all $\alpha \in \Sigma_0$, and the final clause is modelled by $(\{(1, \alpha), (2, \overline{\alpha})\}, \tau, |, \emptyset, \{(1, 1), (2, 2)\}, \emptyset)$ for all $\alpha \in \Sigma_0$.

Note how similar these are to the rules for $\underset{X}{\parallel}$ quoted earlier. The main structural difference is that both sorts of rule apply to *all* visible events, rather than being partitioned by X.

The following presentation of | in CSP is much simpler than the simulation produced by the [20] machine. Extend Σ_0 by a separate copy $\Sigma_1 = \{\alpha' \mid \alpha \in \Sigma_0\}$. Let *IP* (Identity plus Prime) be the renaming that maps $\alpha \in \Sigma_0$ to both α and α', and let *IDP* (Identity plus Dual Prime) map each such α to α and $\overline{\alpha}'$. Then $P \mid Q$ is equivalent to the CSP construct defined

$$P \mid_{ccs} Q = (IP(P) \underset{\Sigma_1}{\parallel} IDP(Q)) \setminus \Sigma_1$$

in the sense that the two processes are strongly bisimilar.

We can therefore conclude that, except for +, CCS is CSP-like.

It is possible to simulate the whole of CCS in CSP, but in a slightly more complex way that does not imply full compositionality over CSP models or a straightforward theory of refinement. An elementary way of doing this is to replace the event τ by a visible analogue (say *tau* as we have seen elsewhere in this paper), and produce models of CCS operators that model the syntax $\tau.P$ by $tau \rightarrow P$. Finally, at the outermost level (in common with the implementation of generalised relabelling given in the previous section) we would hide *tau*. Thus the model of a closed piece P of CCS syntax would be the CSP term $P' \setminus \{tau\}$, where P' is the syntax of P with all operators replaced by their CSP analogues. In this model the analogue of + would be □, since *tau* does resolve □, and the model of parallel would be as above (noting that *tau* is not synchronised) except that the outer hiding $\setminus \Sigma_1$ would be replaced by the renaming that sends all members of Σ_1 to *tau*.

This provides a way of calculating the operational semantics of a CCS term using those of CSP, and also an easy method for using CSP tools such as FDR on such terms.

It is not, in fact, necessary to leave *all* the *tau* actions visible as we build up a term, only those that might resolve a + operator for which our present process becomes (directly or indirectly) an argument. A little structural analysis shows that the only relevant *tau*s are those that are the *first* action that our process performs. It follows that if we apply the following generalised relabelling (standing for "Hide Delayed Taus"), and presented in the relational form discussed earlier, to any term as we build up P', the final result is not affected:

$$HDT = \{(a, s, f(s, a)) \mid a \in \Sigma_0 \wedge s \in \Sigma_0^*\}$$
$$\text{where } f(tau, \langle\rangle) = \tau;\ \ f(a, s) = a \text{ otherwise.}$$

This operator might well be useful if one wants to apply CSP model compressions such as those of FDR [22] in a hierarchical way to CCS constructs, as might the simpler observation that it is always safe to hide any *tau* at a level in the syntax above any +.

This translation will allow any finite-state CCS process to be checked on FDR against the types of specification that FDR supports.

An obvious question then arises: can one model CSP in CCS? The immediate answer to this is "no" since CCS cannot model the multi-way synchronisations permitted by CCS: as soon as two events are synchronised in CCS they are hidden. Another consequence of this is that it is seemingly impossible, in CCS, to implement the style of generalised relabelling discussed in the previous section for the same reason: synchronising P and Reg would hide the event. Of course there may be further interesting questions to ask here about subsets of CSP or extensions of CCS.

5 The π-calculus

The π-calculus [12, 11, 23] builds on the notation of CCS by adding the concepts of name binding and name passing into the language. Like CCS, it does not need process alphabets to define parallel, and therefore the way it expresses mobility is more implicit than one in which passing a label along a channel explicitly changes the alphabets at the two ends, as seems natural for a direct mobile extension of CSP.

The following syntax for the π-calculus is taken from [23]:

PREFIXES $\pi ::= \overline{x}y \mid x(z) \mid \tau \mid [x = y]\pi$

PROCESSES $P ::= S \mid P|P \mid \boldsymbol{\nu} z\ P \mid !P$

SUMMATIONS $S ::= \mathbf{0} \mid \pi.P \mid S + S'$

Here, $\overline{x}y$ represents the sending of the name y via x (akin to the CSP action $x!y$) and $x(z)$ is a construct binding z that represents the receipt of z over x (akin to $x?z$). Like CCS and unlike CSP it has a construct representing the explicit introduction of τ, and one can guard actions with the assertion that pairs of names are the same.

The process constructs are the same as in CCS except that the infinite replication $!P$ (equivalent to $P \mid !P$) takes the place of recursion. The effect of having summations in a separate syntactic class is to allow $+$ only to appear in restricted contexts (similar to the "guarded choice" construct introduced as a precursor to $\Box$ in CSP in both of [6, 16]). Some presentations of the π-calculus omit $+$. Others generalise replication to full recursion $\mu\, p.P$, and we shall follow this latter school. We therefore add $\mu\, p.P$ and p to the syntax of PROCESSES, where p represents a process identifier.

The restriction on the use of $+$ imposed by this syntax is crucial in allowing us to give π-calculus semantics in CSP. When giving semantics to summations we will still need to leave τ visible as tau; but for the main semantics (of processes) this is not necessary. We will therefore find that this version of π-calculus fits more smoothly and compositionally into the world of CSP than does the standard CCS language where the use of τ guards and $+$ is not restricted to the confined syntax of summations.

In our treatment of π-calculus we assume unless stated otherwise that the set $Names$ is countably infinite with a fixed enumeration $\{n_0, n_1, \ldots\}$. If N is a nonempty subset of names then $\mu(N)$ is the name of least index in N, and when $Names - N$ is nonempty $\gamma(N)$ denotes $\mu(Names - N)$. The visible events communicated by π-calculus processes, and making up Σ_π (which plays the role of Σ_0 for our treatment of π-calculus) are of two forms: $x.y$ represents the input of name y over the channel represented by name x, and $\overline{x}.y$ represents the corresponding output of y over x.

The theory of the π-calculus is complex. This is less because processes can pass names around and use them as channels as it is because of the way new names are "created" fresh and distinct from all others. In other words, when two names appear in either nested or separate scopes they are assumed to be different. One has to emphasise the latter because it is possible for a label to be exported by communication *outside* its original scope and therefore visible to the outside world: $\boldsymbol{\nu} v\, P$ creates a *fresh* name, say n, to bind to v, while *restricting* this name from being used illegitimately by P for communicating with the outside world. n cannot be used as a channel into or out of P *until* it has been extruded from this scope by output along another channel.

Our goal is to find a way of dealing with this in CSP in a way that implements the above policy successfully and does not create any artificial distinctions between processes on the basis of exactly which fresh names they extrude.

We now identify two choices that have to be made when constructing our π-calculus semantics.

(1) We can ensure that no artificial distinctions arise thanks to the precise choice of names in two ways.

- The first is to create processes such that, whenever a name is extruded, it is picked nondeterministically from all permitted ones. We will call this the *nondeterministic* approach.
- The second is to ensure that each fresh name that emerges from a process is completely predictable based on knowledge of the preceding trace. We will implement this by ensuring that the name is always the one of least index that might legitimately appear, and call this the *standardised* approach.

(2) One of the trickiest problems in giving the semantics is to ensure that names input by a process from the environment do not get confused with a name that may already exist in the process's scope without having been extruded. It is a basic assumption that these names *are* different, so how do we achieve this?

- One approach is to allow the environment, at any time, to output any name at all to the process. The semantics then has to perform an operation related to α conversion on any fresh and unextruded name it holds that clashes. We will call this the *unified* approach.
- The opposite of this, which we will call the *bipartite* approach, is to split the fresh name space into two infinite parts and allocate one to the process and the other to the environment. In other words the process will simply assume at all times that the environment will not output any name to it that is in that part of the process's fresh name space unless that name has previously been extruded by the process.

In this paper we will first consider the unified name space approach, providing an initial translation into CSP followed by constructs that respectively map this into the nondeterministic and standardised approaches. We then summarise the differences in how these steps are taken in the bipartite approach.

We would expect any semantics for π-calculus to have complete symmetry in those names that the environment generates as fresh (though in the unified approach the symmetries will be more complex since they will need to factor in the fact that the names chosen by the environment affect the values of the names chosen subsequently by the process). We would expect any nondeterministic semantics to have complete symmetry in the names chosen by the process.

5.1 Preserving inequality

The π-calculus is fairly straightforward to translate into our CSP extended by $|_{ccs}$, with the exception of its handling of freshness. We first show how to translate summations. If S is a SUMMATION then $\mathrm{CSP}[S]_+$ will be a CSP term in which τ actions remain visible as *tau*. The corresponding semantics for a PROCESS P is written $\mathrm{CSP}[P]$.

- $\mathrm{CSP}[0]_+ = STOP$
- $\mathrm{CSP}[[x = y]\pi.P]_+ = \mathrm{CSP}[\pi.P] \lhd x = y \rhd STOP$
- $\mathrm{CSP}[\tau.P]_+ = tau \rightarrow \mathrm{CSP}[P]$
- $\mathrm{CSP}[x(y).P]_+ = x?z \rightarrow \mathrm{CSP}[P[z/y]]$
- $\mathrm{CSP}[\overline{x}y.P]_+ = \overline{x}.y \rightarrow \mathrm{CSP}[P]$
- $\mathrm{CSP}[P + Q]_+ = \mathrm{CSP}[P]_+ \,\square\, \mathrm{CSP}[Q]_+$

Here $P \lhd b \rhd Q$ is Hoare's infix representation conditional choice: it equals P if b it *true* and Q if b is *false*.

We can interpret a summation as a process via hiding, and both process identifiers and recursion translate directly into CSP:

- $\mathrm{CSP}[P] = \mathrm{CSP}[P]_+ \setminus \{tau\}$
- $\mathrm{CSP}[p] = p$
- $\mathrm{CSP}[\mu\, p.P] = \mu\, p.\mathrm{CSP}[P]$

The remaining constructs are parallel $P \mid Q$ and restriction $\boldsymbol{\nu} z\, P$. These are more difficult because of the way they handle fresh names. The effects we want to achieve are set out below.

- $\boldsymbol{\nu} z\, P$ creates a fresh name z' for the placeholder z that is used within P. This is different from all other names known to P at the point of creation and all names that P sees subsequently that cannot result from other processes reflecting z' back to P.
- $\boldsymbol{\nu} z\, P$ may *extrude* this z' from this scope by means of output $\overline{x}z'$ on some other channel (i.e. $x \neq z'$).
- After this extrusion, $\boldsymbol{\nu} z\, P$ may use the name z' as a channel name, output it again, etc. However before the extrusion any of P's communications that use it as a channel name are blocked. This by analogy with CCS restriction, explains why this operator is called "restriction".
- In the parallel composition $P \mid Q$, the processes P and Q never extrude the same fresh name.
- Because of the way in which interactions between P and Q are hidden, P may extrude a fresh name z to Q or vice-versa, without the external environment seeing z. This expansion of the scope of z is restricted as above: it may not use the name z in a visible way until z has been extruded *from the parallel composition* via output on a different channel.
- Each fresh name extruded from $P \mid Q$ must be different from all names either P or Q knew originally or subsequently input.

The above must hold for any strategy for assigning and managing the fresh names created within a term P. We use the term "managing" here because there is no sensible way, within the unified approach to name choice, of ensuring that once a fresh value z has been created but not yet extruded, the external environment does not independently invent the same name (we will call this a name collision). If this happened it would cause confusion both in the environment and the process P.

This is why we need to use relabelling. In the instance above, one thing we could do would be to pick a further fresh name z' not known to P, and apply the renaming $[\![{}^{z,\,z'}/z',z]\!]$ (transposing the two names) to P from the point where the environment communicates z to it. Thus P will see z as z' (correctly, a value that is fresh to it and distinct from the z it already knows about) and the environment will, if and when P ultimately extrudes z the outside world will see it as z' (correctly, a value it has not seen before). Since neither z not z' has appeared in the trace before this point, each of these names only has a single role in the whole trace. We will use relabellings such as this to avoid collisions both for $\boldsymbol{\nu} z\, P$ and $P \mid Q$, the latter because of the scope expansion issues discussed above.

We will introduce a relabelling called $OF(N, K, P)$ ("output first") which, given a set of names that the process P might extrude, and a set that it knows initially, introduces a transposition of the above form each time the process inputs from the environment a name that clashes with a member of N. The set N may diminish as the process evolves since some of the names in it may be extruded; and K may increase as P learns more names.

In order to keep track of the transpositions that are introduced as the system evolves, we need to introduce a parameter ξ that is a bijection on the process alphabet Σ_π. At any time, ξ will be the the function that maps names as seen on the inside of the relabelling to the corresponding names seen by the environment. While this function will evolve as the system progresses, as soon as a name has been seen in a trace of P its image remains fixed thereafter. It follows that at the end of a trace ξ is a permutation on *Names* that translates P's view to the environment's view of every member in the trace. Initially, ξ is the identity function.

The most straightforward way of presenting OF is by defining a Reg_{OF} process that creates the relabelling using the construction $(*)$.

In the following we assume that K (the set of names representing "common knowledge"[3] between process and environment) and N (the set of potentially fresh names yet to emerge from the process) are disjoint. $\mathit{Reg}_{OF}(K, N, \xi)$ takes the form

$$\Box\{(a, b) \to \mathit{Reg}_{OF}(K', N', \xi') \mid (a, b, K', N', \xi') \in C\}$$

where a is the process P's event, b is the environment's view of the same event, and C is a set of clauses that we will describe below, each representing a different sort of event that the relabelling allows.

We consider the cases of output $\overline{x}.y$ and input $x.y$ events separately, splitting these depending on which of K and N y belongs to. In both cases we restrict x to be K: it is part of the role of OF to prevent P from using names in N before these have been extruded from scope; and if our semantics are sensible P could never use a channel name that is outside $K \cup N$.

We now enumerate the various clauses of $\mathit{Reg}_{OF}(K, N, \xi)$:

- If both the channel and data are part of common knowledge, then an input does not change the parameters:

$$\{(x.y, \xi(x).\xi(y)) \to \mathit{Reg}_{OF}(K, N, \xi) \mid x, y \in K\}$$

[3] K is the set of names that P sees as known by both itself and its environment; $\xi(K)$ is the environment's view of this set.

- If both the channel and data are part of common knowledge, then an output does not change the parameters:

$$\{(\overline{x}.y, \overline{\xi(x)}.\xi(y)) \rightarrow \mathit{Reg}_{OF}(K, N, \xi) \mid x, y \in K\}$$

- If P outputs a name in N, then this is removed from N and added to K

$$\{(\overline{x}.y, \overline{\xi(x)}.\xi(y)) \rightarrow \mathit{Reg}_{OF}(K \cup \{x\}, N - \{x\}, \xi) \mid x \in K, y \in N\}$$

- If P inputs a name outside $K \cup N$, then this is equivalent to the environment extruding a name to P, so it is added to the common knowledge:

$$\{(x.y, \xi(x).\xi(y)) \rightarrow \mathit{Reg}_{OF}(K \cup \{x\}, N, \xi) \mid x \in K, y \notin N \cup K\}$$

- Finally, if P were to input a name y in N, then this is a name collision, since the environment has yet to be told of N by P. However, in the unified approach, there is nothing to stop the environment inventing such a name independently. Therefore the mapping ξ is changed so that the new name maps outside $K \cup N$ under ξ^{-1}. The simplest way to do this is to transpose x and the name $\gamma(K \cup N)$, which we recall is the one of least index not in this set. We write this transposition (which maps all other names to themselves) as $xp(x, \gamma(K \cup N))$. The clauses of this type are thus

$$\{(x.n, \xi(x).\xi(y)) \rightarrow \mathit{Reg}_{OF}(K \cup \{n\}, N, \xi \circ xp(x, n)) \mid x \in K, y \in N, n = \gamma(K \cup N)\}$$

Notice that all of the above English descriptions are formulated from the point of view of P, with the already established permutation ξ assumed.

It is these transpositions in OF that represent the analogy of α-conversion that we discussed earlier. Both of these things have the role of avoiding clashes between external and bound identifiers. Whereas traditional α-conversion does not change the semantics of a term, our transpositions do, but this is only because “bound” identifiers in π-calculus can, unusually, be seen from the outside.

This relabelling is deterministic since there is only one way $\mathit{Reg}_{OF}(K, N, \xi)$ can perform any given action, and clearly it has no τ-generated nondeterminism. Indeed, since there is only one action that $\mathit{Reg}_{OF}(K, N, \xi)$ can perform for any given action seen by the environment, $OF(N, K, \cdot)$ could be presented as a relabelling generated by a relation between P's visible actions, the environment's visible actions and the environment's traces. But since it does evolve naturally a step at a time, the above presentation is probably the best.

We are now very close to being able to give a semantics to $\boldsymbol{\nu} z\, P$, but before we do that we need to establish what the right values are to use for K and N when we come to use OF. We will also need a semantic mechanism to keep the fresh names invented by two parallel processes distinct.

The best way to do these things is to add parameters κ and σ to the semantics so they become $\textsc{csp}[S]_{+}\kappa\sigma$ and $\textsc{csp}[P]\kappa\sigma$. κ will represent the initial common knowledge of names by P and its environment or context: a finite set of names that includes all free names in P. σ will be an infinite set of names, disjoint from κ, that are available to be used as fresh. So for example we will now have $\textsc{csp}[P + Q]_{+}\kappa\sigma = \textsc{csp}[P]_{+}\kappa\sigma \,\Box\, \textsc{csp}[Q]_{+}\kappa\sigma$ and (the only clause given to date with a significant change) $\textsc{csp}[x(y).P]_{+}\kappa\sigma = x?y \rightarrow \textsc{csp}[P]_{+}(\kappa \cup \{y\})(\sigma - \{y\})$.

We can then write

$$\textsc{csp}[\boldsymbol{\nu} z\, P]\kappa\sigma = OF(\{\gamma(\sigma)\},\ \kappa,\ \textsc{csp}[P[\gamma(\sigma)/z]](\kappa \cup \{\gamma(\sigma)\})(\sigma - \{y\}))$$

In other words, the name of least index not in κ is chosen to bind to z, and this name is then "protected" by the OF operator, which also prevents it from being used as a channel by P until it has been output.

The interactions between P and Q in $P \mid Q$ are calculated in the same way as in CCS. It follows that the same CSP construction we used in the last section can be used here provided we sort out what happens to the sets of names that P and Q use and generate, and provided we ensure that $P \mid Q$ obeys the rules discussed above for it being an expanded scope for some of these generated names.

The parameter σ gives us the ability to keep the names that P and Q generate distinct from each other. To do this we assume that we have functions Π_1 for $i = 1, 2, 3$ such that, for any infinite set σ of names, $\{\Pi_1(\sigma), \Pi_2(\sigma), \Pi_3(\sigma)\}$ partitions σ into three infinite sets. [One such triple of functions would allocate the names of increasing index in σ to the three sets in a round robin fashion.] In evaluating $\textsc{csp}[P \mid Q]\kappa\sigma$ these three sets will respectively represent the sets of fresh names that might be generated by P, by Q, and outside this system.

We can then define

$$\textsc{csp}[P \mid Q]\kappa\sigma = OF(\Pi_1(\sigma) \cup \Pi_2(\sigma),\ \kappa,\ \textsc{csp}[P]\kappa(\Pi_1(\sigma)) \mid_{ccs} \textsc{csp}[Q]\kappa(\Pi_2(\sigma)))$$

The role of OF here is to protect any fresh names extruded from P or Q into $P \mid Q$ but not yet to the environment. Since it does not know exactly which fresh names have been extruded in this way, it protects the whole of the set $\Pi_1(\sigma) \cup \Pi_2(\sigma)$. Here, of course $\mid_{ccs}$ is the CSP translation given earlier of the CCS operator $\mid$.

Note that as far as P's and Q's choices of names is concerned, we have used what amounts to the bipartite approach by ensuring that they choose from disjoint sets.

5.2 Nondeterministic fresh names: a lesson on CSP models

We might say that the above clauses give a *provisional* semantics, since they suffer from the lack of abstraction discussed above caused by specific choices of fresh names.

In many ways the most elegant approach to this problem is to apply a mapping that in effect maps each assignment of fresh names to a representation of its symmetry class: the processes that might have been obtained under a different strategy for assigning fresh names.

We can attempt to do this by identifying each semantic value $\textsc{csp}[P]\kappa\sigma$ with the nondeterministic composition of the set of its values under renamings that change the choices of fresh names it picks from σ. The following operator is perhaps the most natural way to do this

$$NFN(P, \sigma) = \bigsqcap\{P[\![\xi \cup id_{(Names-\sigma)}]\!] \mid \xi : \sigma \to \sigma, \xi \text{ bijection}\} \quad (\dagger)$$

One can gain much insight into CSP models and the way to use them correctly for π-calculus by analysing whether this approach works or not.

We will now show that there are pairs of processes P and Q that we would like to regard as semantically equivalent for which there is no such ξ with $\textsc{csp}[P]\kappa\sigma[\![\xi]\!] = \textsc{csp}[Q]\kappa\sigma$.

If we have a process that has two different behaviours on which it extrudes a free name, it should not matter whether these two names are the same or different. Consider, for example, $P = \tau.(\boldsymbol{\nu} z\, \overline{x}z.0) + \tau.(\boldsymbol{\nu} z\, \overline{y}z.0)$ (where the above semantics will always output the same fresh name via whichever of $\overline{x}$ and $\overline{y}$ the environment chooses) and $Q = \tau.(\boldsymbol{\nu} z\, \overline{x}z.0) + \tau.(0 \mid$

$(\boldsymbol{\nu} z\, \overline{y}z.0))$ (where, depending on Π_2, it may not). No renaming of the above form can map P to Q. For similar reasons the CSP operational semantics of P and Q with construct (†) applied to them are not bisimilar, since no bijection ξ can map the state P to the state Q.

Thus the *initial* choice of a single permutation does not characterise semantic equivalence in a model where processes are represented as tree structures.

One of the main differences in CSP-style models based on *linear* observations of processes is that they deliberately obscure when choices are made: hence the CSP distributivity law $a \rightarrow (R \sqcap S) = (a \rightarrow R) \sqcap (a \rightarrow S)$, for example. A CSP-style semantics of the processes P and Q above will only let you look at things that occur down a linear observation: it will not let you examine the consequences of outputting on channels $\overline{x}$ and $\overline{y}$ in the same observation; only in two separate ones. It should not therefore not matter, in this style of model, if the choice of a permutation ξ is made initially or in stages as long as every name that actually appears is mapped to the same place as in ξ.

We would expect such a model of $\text{CSP}[P]\kappa\sigma$ still to output the same name along these two channels and $\text{CSP}[Q]\kappa\sigma$ probably to output different ones, so the provisional semantics of P and Q will still be different even if we only record process traces. However, when we look at the effect on their traces of the (†) construct, these two values are mapped to the same value. The point is that any trace of $\text{CSP}[P]\kappa\sigma$ will be the result of some ξ being applied to a trace of $\text{CSP}[Q]\kappa\sigma$, and vice-versa. The fact that *different* ξ may be needed for different traces is immaterial: every trace of $\text{CSP}[P]\kappa\sigma$ will be a trace of $\mathit{NFN}(\text{CSP}[Q]\kappa\sigma, \sigma)$ and vice-versa.

We can conclude that, at least for the finite traces semantics of CSP, construction (†) gives the provisional CSP semantics the necessary abstraction. There are subtle problems, however, when we come to study types of behaviour that appear in other CSP models. These are, on the one hand, refusal and acceptance sets and, on the other, behaviours recording infinite traces. To avoid this second sort of difficulty, let us assume for the time being that we are interpreting CSP over a model that does not have infinite traces.

Most CSP models have refusal or acceptance sets to enable us to detect things like deadlock. These can lead to undesirable semantic differences between π-calculus terms in two related ways. As an example, consider the processes

$$\boldsymbol{\nu} y\, \overline{x}y.\boldsymbol{\nu} z.\overline{x}z.0 \qquad (\boldsymbol{\nu} y\, \overline{x}y.0) \mid (\boldsymbol{\nu} z\, \overline{x}z.0)$$

Each of then simply extrudes two fresh names in succession along $\overline{x}$, and they ought to be regarded as equivalent.

With conventional failures-based models, in which refusal sets are sets of *events*, these two processes are not equivalent under the *NFN* mapping. For the left-hand process will have failures of the form $(\langle\rangle, \Sigma - \{\overline{x}.n\})$, while the largest refusal sets on $\langle\rangle$ of the right-hand process will omit $\overline{x}.n$ and $\overline{x}.m$ for two different names n and m: in effect the parallel structure gives the environment a "choice" of two different names.

A further difficulty arises in more elaborate CSP models such as *refusal testing* where it is possible to see the refusal or acceptance of an event that extrudes a name several steps before it actually appears in the trace. Such a name might or might not get relabelled between these two appearances, which can again lead to undesired inequivalences as well as some confusing-looking observed behaviours.

There is a simple solution to both these problems: in languages such as π-calculus where all communication between processes happens over point-to-point channels, and processes inputting on a channel always do so *non-selectively* – they cannot accept some communications on it but refuse others – we get a better model of refusal sets and acceptance sets by constructing them solely of channel names, not events. This was recognised for occam in [14].

It will therefore be impossible to tell, in any "channel-based" CSP-style model, between the sequential and parallel processes above that extrude fresh names, to see any fresh name in a recorded behaviour before it is extruded, or indeed to make the same sort of distinctions based on non-fresh names. Thus, for example, $\overline{x}y.P + \overline{x}z.Q$ and $\tau.\overline{x}y.P + \overline{x}z.Q$ will be identified as processes (though not as summations) in any such model.

Given any of these channel-based CSP models $\mathcal{M}$ we can therefore give a proper semantics to the π-calculus. The semantics for a summation and a process will be written $\mathcal{M}^1_\sqcap[\![S]\!]_+\kappa\sigma$ and $\mathcal{M}^1_\sqcap[\![P]\!]\kappa\sigma$ which are defined to be the respective CSP interpretations over $\mathcal{M}$ of the terms:

$$\mathit{NFN}(\mathrm{CSP}[S]_+\kappa\sigma, \sigma) \quad \text{and} \quad \mathit{NFN}(\mathrm{CSP}[P]\kappa\sigma, \sigma)$$

The superscript 1 here means that this is a semantics using a single (unified) name space, and $\sqcap$ means that this is the *nondeterministic* semantics. The alternatives for these are 2 (bipartite) and μ (standardised).

These can readily be turned into denotational semantics. The preliminary semantics $\mathrm{CSP}[P]\kappa\sigma$ can already be interpreted in this way if we add an extra parameter ρ: an environment that maps process identifiers to functions from common knowledge sets κ and fresh-variable sets σ to $\mathcal{M}$.[4] They will thus be written $\mathcal{M}^1_\sqcap[\![P]\!]\rho\kappa\sigma$. Thus the semantics of a process variable is given by $\rho(p)\kappa\sigma$. The semantics of each non-recursive operator is then just the interpretation over $\mathcal{M}$ of the CSP syntax into which we have already translated it, applied to sub-terms as appropriate. The semantics of recursion is just the same fixed-point calculation that is appropriate for $\mathcal{M}$: sometimes a refinement-least fixed point, sometimes a subset-least fixed point, and sometimes something more complex as in [17].

The full denotational semantics is then obtained in the same way except that for some syntactic forms (parallel, restriction) it is necessary to apply the operator $\mathit{NFN}(\cdot, \sigma)$ (interpreted as an operator over $\mathcal{M}$) to the result.

$\mathit{NFN}(\cdot, \sigma)$ was presented above as the nondeterministic choice of a set of renamings. It is easy to reformulate it as a nondeterministic relabelling, if desired.

Since no name other than a member of $\mathit{fn}(P)$ means anything special to a π-calculus term, we must expect that, for any permutation η of $\mathit{Names} - \mathit{fn}(P)$, the processes P and $P[\![\eta]\!]$ are equivalent. Since *we* give meaning to members of $\kappa - \mathit{fn}(P)$ and σ, we cannot expect that $\mathcal{M}^1_\sqcap[\![P]\!]\kappa\sigma = \mathcal{M}^1_\sqcap[\![P]\!]\kappa\sigma[\![\eta]\!]$ in general, but we can expect that

$$\mathcal{M}^1_\sqcap[\![P]\!]\rho\kappa\sigma = \mathcal{M}^1_\sqcap[\![P]\!]\rho(\eta^{-1}(\kappa))(\eta^{-1}(\sigma))[\![\eta]\!]$$

The infinite nondeterministic choice used in NFN actually goes beyond the CSP syntax that some of the standard models can handle, specifically those which, like the failures-divergences model $\mathcal{N}$, have representations of divergence but not other infinite behaviours such as infinite traces. The reason for this limitation is that when a process has unbounded nondeterminism it is not possible to infer whether $P \setminus X$ can diverge from its finite traces

[4] These parameters are required because both the common knowledge and availability of fresh names may well have changed at the point of call, and it seems to be correct to evaluate the recursive call in that new world, just as is done by the textual substitution of term re-writing. In the spirit of a pure denotational semantics, fully espousing the traditions of [24], we might well wish to discriminate between *Names* and the identifiers *Ide* that denote them: this would allow definitions of input and restriction without syntactic substitution.

alone. Hiding appears in our CSP translation of π-calculus as part of the definition of $|_{ccs}$. Fortunately, however, the symmetry of π-calculus semantics under permutations on names means that whenever there are arbitrarily long finite traces of terms P and Q that combine under $|$ to give prefixes of a fixed finite trace, then there is also a pair of infinite traces with the same property.[5] It follows that whenever there are arbitrarily long finite traces created by the part of the CSP construction of $|_{csp}$ other than hiding, that the hiding maps to prefixes of a given trace s, then (i) there is an infinite trace of the same process that hides to a prefix of s and (ii) s is itself recorded as a divergence by divergence strictness.

It is therefore not necessary to use CSP models involving infinite traces if one wishes to handle strict divergence accurately in π-calculus. If, however, we want to calculate the infinite traces for other reasons or to handle non-strict divergence using the techniques developed in [17], we need to use a model that represents them explicitly. As we said above, this brings with it the danger of distinguishing processes that we would naturally hope to be equivalent: consider the following pair of processes:

$$\mu\, p.\boldsymbol{\nu} z\, \overline{x}z.p \quad \text{and} \quad \mu\, p.\boldsymbol{\nu} y\, \boldsymbol{\nu} z\, \overline{x}z.p$$

These both output an infinite supply of fresh names over channel $\overline{x}$. There is every reason for wanting to identify them, and indeed the terms P and $\boldsymbol{\nu} y\, P$ when y is not free in P. A little thought, however, will reveal that our $\textsc{csp}[\cdot]$ semantics will have $\mu\, p.\boldsymbol{\nu} z\, \overline{x}z.p$ output all the enumerated fresh names one by one, whereas $\mu\, p.\boldsymbol{\nu} y\, \boldsymbol{\nu} z\, \overline{x}z.p$ will only output every other one.

For finite traces, this problem is easily remedied by an appropriate permutation η: any injective finite partial function from a set to itself can be extended to a permutation. This does not work for infinite traces: once we map the single infinite trace of the right-hand process to the single infinite trace of the left-hand one, there is nowhere left to map all the names that the right-hand process has missed out. We must therefore conclude that the semantics we have built to date, if interpreted over a model with infinite traces, makes undesirable distinctions between processes.

We can also conclude that the semantic values it creates are not always *closed*, in the sense that if every prefix of an infinite trace is present then so it the infinite trace itself. This is not in itself worrying, but it means that the nondeterministic interpretation of π-calculus provides the only example known to the author of a semantics involving hiding in which models like $\mathcal{N}$ work accurately despite the processes not being closed!

[5] We can deduce this as follows. Consider the operational semantics of P and Q derived from $\textsc{csp}[P]\kappa\sigma_P$ and $\textsc{csp}[Q]\kappa\sigma_Q$ (without any application of NFN, where σ_P and σ_Q are the sets assigned to them by the $\textsc{csp}[\cdot]\kappa\sigma$ semantics of $|$). These are finite branching, in the sense that every finite sequence of visible events and τs can only lead to finitely many distinct states. This depends on the fact that no single state of P or Q can have more than a finite number of distinct output $\overline{x}.y$ actions available, for otherwise $|_{ccs}$ might introduce infinite branching on τ.

The existence of arbitrarily long pairs of finite traces of P and Q such that $P \mid Q$, under perhaps different ξs, can give rise to prefixes of a given finite trace s means that $\textsc{csp}[P \mid Q]\kappa\sigma$ can itself behave in the same way except that the names of the fresh names extruded might be different. So if we examine that part of the operational semantic tree of $\textsc{csp}[P \mid Q]\kappa\sigma$ in which the visible trace is a prefix of s with modifications to the names of extruded fresh names permitted, it is infinite and therefore, by König's lemma, has an infinite path that necessarily ends (as viewed from the outside) in an infinite sequence of τs. It follows that $\textsc{csp}[P \mid Q]\sigma\kappa$ has a divergence that is a prefix of s except that the finitely many extruded names might be different. Since there is certainly a permutation η that maps these names to the ones seen in s, it follows that the corresponding exact prefix of s, and hence s itself from divergence strictness, are divergences of $\mathit{NFN}(\textsc{csp}[P \mid Q]\kappa\sigma, \sigma)$.

An easy, if perhaps extreme, way of solving this problem is to move to having an uncountable set of *Names*; the point being that such a set can never be used up, even in an infinite trace. It is difficult to see how we can otherwise solve it in the nondeterministic spirit of *NFN*, at least in the unified case, when one considers how the environment is also allowed to pick an infinity of fresh names in some infinite traces.

To avoid these difficulties and more complex arguments later in this paper, in this paper's initial treatment of π-calculus via CSP we will only consider CSP models where each recorded behaviour only involves a finite number of names in each trace: ones that are relational images of either $\mathcal{FL}$ (as described earlier) or $\mathcal{FL}^{\Downarrow}$ (with strict divergence but without infinite behaviours), in their channel-based versions where all the acceptance sets consist only of channel names.

The fact that a π-calculus term only "knows" a finite set of names immediately, coupled with the way in which we are using channel-based models, means that all these acceptance sets are themselves finite. It follows that only finitely many members of *Names* appear in any behaviour recorded in $\mathcal{FL}$ or $\mathcal{FL}^{\Downarrow}$. We will use this fact repeatedly. We will refer to these as *finite behaviour models*.

This restriction allows one immediate simplification. The nondeterministic choice in the definition of *NFN* is over an uncountable set of functions. Over a model in which only finitely many names appear in a trace it is equivalent to additionally restrict the set of bijections to those which, except for a finite set of names, are equivalent to the identity. This is now a countable choice.

5.3 Standardised names

The CSP semantics that emerges from the nondeterministic approach given above has much to recommend it, not least the symmetry. It can, however, be viewed as unnecessarily infinitary, depending as it does on infinite nondeterministic choice.

We can solve this problem by removing from the process all opportunity to choose the fresh names it extrudes. We ensure that each name to appear in this way is always the least allowed: the member of σ with least index that has not already appeared in the trace. We can achieve this with a relabelling $SFN(P, K, S)$ (standardised fresh names) presented in a similar style to OF, namely via its regulator process $Reg_{SFN}(K, S, \xi)$. This is, like Reg_{OF}, formed by composing a sets of clauses with $\Box$. The parameters are similar to those used for Reg_{OF}, except that this time the parameters are based mainly on the external view of the system since it is that which we standardising.

K (initially κ) is the external view of the current common knowledge.

S (initially σ) is the external view of the set of free names that are available to be extruded by the system.

ξ is a partial function, with domain K, that maps the external view of common knowledge to the internal one. Initially this is the identity function on κ.

The clauses forming Reg_{SFN} are:

- If both the channel and data are part of common knowledge, then an input does not change the parameters:

$$\{(\xi(x).\xi(y), x.y) \to Reg_{SFN}(K, N, \xi) \mid x, y \in K\}$$

– If both the channel and data are part of common knowledge, then an output does not change the parameters:

$$\{(\overline{\xi(x)}.\xi(y), \overline{x}.y) \rightarrow \mathit{Reg}_{\mathit{SFN}}(K, N, \xi) \mid x, y \in K\}$$

– If P outputs a fresh name x (one not in $\mathit{range}(\xi) = \xi(K)$), then this is seen on the outside as the least index member of S.

$$\{(\overline{\xi(x)}.y, \overline{x}.\,\mu(S)) \rightarrow \mathit{Reg}_{\mathit{SFN}}(K \cup \{\mu(S)\}, S - \{\mu(S)\}, \xi + [\mu(S) \mapsto y] \mid x \in K, y \notin \xi(K)\}$$

– If P inputs a fresh name, then we standardise this so that it is always the one of least index not in K.

$$\{(\xi(x).\gamma(\xi(K)), x.y) \rightarrow \mathit{Reg}_{\mathit{SFN}}(K \cup \{y\}\}, S - \{y\}, \xi + [y \mapsto \gamma(\xi(K))] \mid x \in K, y \notin K\}$$

The way we formulated this last clause means that we have not only standardised the way in which fresh names are extruded from P, but also the names that P inputs as fresh from the environment. The latter is not strictly necessary to achieve the standardised external view of our process, but (i) it has a pleasing symmetry and (ii) might well make for more efficient automated verification.

As with *NFN*, we can add this new operator to the CSP translation

$$\mathcal{S}[\![P]\!] = (\textsc{csp}[P](\mathit{fn}(P))(\mathit{Names} - \mathit{fn}(P)))\langle\!\langle \mathit{SFN}(\mathit{fn}(P), \mathit{Names} - (\mathit{fn}(P)))\rangle\!\rangle$$

where $\mathit{fn}(P)$ is the set of free names in P, or construct a denotational semantics based on any channel-based CSP model if we add an environment of the same form as used in the nondeterministic semantics.

We have the following interesting property of the relabellings used in our two semantics:

Lemma 1. *This result is about CSP processes defined over the alphabet Σ_π which respect the π-calculus discipline that no name not in the initial set K is used as a channel before it is input or output over another channel, and such that the names it thus "creates" through output are confined to the infinite set S disjoint from K.*

For any such process:

(i) For any permutation ξ of names that is the identity on $\mathit{Names} - S$, $P\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle = P[\![\xi]\!]\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle$ holds up to strong bisimulation.

(ii) $P\langle\!\langle \mathit{SFN}(K,S\rangle\!\rangle) = \mathit{NFN}(P,S)\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle$ holds in every finite-behaviour CSP model.

The first part simply says that it does not matter how the fresh names generated by P are permuted if they are to be renamed into standard order. The second part then follows because in any such model the $\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle$ operator is distributive over nondeterministic choice.

This lemma implies that the equivalence induced by the nondeterministic semantics is at least as fine as that induced by the standardised one.

If we were to allow CSP models recording behaviours with an infinity of names in individual traces, part (ii) of this lemma would not hold in reverse (i.e. with the roles of *NFN* and *SFS* transposed). This is because applying $\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle$ to a behaviour b that extrudes an infinity of *Names* from S always maps it to one in which the *whole* of S appears. This means that $\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle$ will identify pairs of behaviours that no permutation on S

can. This is the same issue we saw when discovering undesirable inequivalences created by the nondeterministic semantics with infinite traces.

If, however, a behaviour b only involves a finite number of names, then the result of applying $\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle$ to it is certain to leave an infinite set of names unrecorded in b. As in the earlier example, it is then possible to extend the finite injective mapping from S to itself, created by $\mathit{Reg}_{\mathit{SFN}}$ by the time it has communicated all the names used in b, to a bijection. This proves the following inverse of the lemma above.

Lemma 2. *Under the same conditions as Lemma 1,*

$$\mathit{NFN}(P,S) = \mathit{NFN}(P\langle\!\langle \mathit{SFN}(K,S)\rangle\!\rangle, S)$$

holds in any channel-based, finite-behaviour CSP model.

We can collect these two results into the following theorem:

Theorem 1. *In any such model channel-based, finite-behaviour model of CSP the standardised and nondeterministic semantics for π-calculus represent the same equivalence over process terms.*

The author believes that in cases, beyond the scope of this paper of CSP, of models including infinite-name behaviours, the standardised semantics will give the correct equivalences. In future discussions in this paper we will concentrate on the standardised form.

The parameters $\kappa \supseteq \mathit{fn}(P)$ and infinite σ disjoint to κ are structurally important to the semantics, but however they are chosen within these constraints they do not change the equivalence induced by the semantics, as is demonstrated by Lemma 4 below. The crucial result needed to establish this is Lemma 3: the role it plays is that it allows us to disentangle the fresh names picked by process and environment.

If b is any finite-name behaviour from a channel-based CSP model, $\mathcal{M}$ let $\mathit{enames}(b,K)$ be the (necessarily finite) set of names first *input from* the environment that are not in the common knowledge set K. We then have the following:

Lemma 3. *If P is a π-calculus process with $\mathit{fn}(P) \subseteq \kappa$, and σ (disjoint from κ) is infinite, then*

$$b \in \mathcal{M}^1_\mu[\![P]\!]\kappa\sigma \Leftrightarrow b \in \mathcal{M}^1_\mu[\![P]\!]\kappa(\sigma - \mathit{enames}(b,K))$$

Lemma 4. *If P and Q are π-calculus processes with $\mathit{fn}(P)\cup\mathit{fn}(Q) \subseteq \kappa\cap\kappa'$, and σ (disjoint from κ) and σ' (disjoint from κ') are infinite, then over any finite-behaviour channel-based CSP model $\mathcal{M}$ we have*

$$\mathcal{M}^1_\mu[\![P]\!]\kappa\sigma = \mathcal{M}^1_\mu[\![Q]\!]\kappa\sigma \Leftrightarrow \mathcal{M}^1_\mu[\![P]\!]\kappa'\sigma' = \mathcal{M}^1_\mu[\![Q]\!]\kappa'\sigma'$$

It follows that we can talk about *the* congruence on π-calculus induced by a given CSP model of this type, rather than needing to calculate it relative to a particular κ and σ. In deciding the equivalence of two processes we can compare their semantics with $\kappa = \mathit{fn}(P) \cup \mathit{fn}(Q)$ and $\sigma = \mathit{Names} - \kappa$. The above lemma demonstrates, *inter alia* that this is an equivalence relation.

5.4 Bipartite semantics

In the above semantics we ensured that a pair of parallel processes $P \mid Q$ never create names that collide with each other. Our main reason for doing this was to ensure that two fresh names output directly to the environment respectively by P and Q do not collide. It has the side-effect, however of meaning that when P inputs a fresh name created by Q or vice-versa they never need to call upon the transpositions of OF to avoid collisions.

The only entity that we have not been able to control enough to prevent collisions is the *external environment* because we have always allowed it to output *any* name to the process we are observing. It is this that characterises the *unified* approach.

In the *bipartite* approach, the set σ or S of names that the process can choose from is not only infinite, but its complement (the names either initially known or available to be created fresh by the environment) is also infinite. We replace the parameter κ, which we will not need with the bipartite treatment of freshness, with a different set, ζ. This represents the set of names which it is legitimate to receive as inputs over channels. Before the process has started to run ζ is always the complement of σ, but at some points in the semantics it is a proper subset of the complement, the difference being the fresh names that the process has created (and so are no longer in σ) but not yet extruded.

Rather than give this alternative semantics in detail, we summarise the differences below. The translations of the input and output constructs are now as follows, noting we are defining a second translation $\textsc{csp}2[\cdots]$:

$$\begin{aligned} \textsc{csp}2[x(y).P]_{+}\zeta\sigma &= x?y : (\mathit{Names} - \zeta) \rightarrow \textsc{csp}2[P]_{+}(\zeta)\sigma \\ \textsc{csp}2[\overline{x}y]_{+}\zeta\sigma &= \overline{x}.y \rightarrow \textsc{csp}2[P]_{+}(\zeta \cup \{y\})\sigma \end{aligned}$$

We get a considerable simplification by being able to drop the most complicated part of the OF relabelling. OF can in fact be replaced by parallel composition with a process $\mathit{BOF}(\zeta)$ (bipartite output first) that enforces the discipline that no name not in ζ can be used as a channel before being output by the process:

$$\begin{aligned} \mathit{BOF}(Z) = \; & \square\{x?y : Z \rightarrow \mathit{BOF}(Z) \mid x \in Z\} \\ & \square \\ & \square\{\overline{x}?y : \mathit{Names} \rightarrow \mathit{BOF}(Z \cup \{y\}) \mid x \in Z\} \end{aligned}$$

The translation of $\boldsymbol{\nu} z\, P$ thus becomes

$$\textsc{csp}2[\boldsymbol{\nu} z\, P]\zeta\sigma = \mathit{BOF}(\zeta) \underset{\Sigma_\pi}{\|} \textsc{csp}2[P[\mu(\sigma)/z]]\zeta(\sigma - \{\mu(\sigma)\})$$

There is not no need to have the function Π_3 in the semantics of $|$: all we need are functions Π'_1 and Π'_2 that partition σ into two disjoint infinite sets, which each process having the other's σ incorporated into its ζ, as are any names in $\mathit{Names} - (\zeta \cup \sigma)$ since these have already been declared at the point the parallel composition is started and we could easily have given the output end of such a channel to P and the input end to Q. $\textsc{csp}2[P \mid Q]\zeta\sigma$ is defined to be

$$\mathit{BOF}(\zeta) \underset{\Sigma_\pi}{\|} (\textsc{csp}2[P](\mathit{Names} - \Pi'_1(\sigma))(\Pi'_1(\sigma)) \mid_{ccs} \textsc{csp}2[Q](\mathit{Names} - \Pi'_2(\sigma))(\Pi'_2(\sigma)))$$

There is one important respect in which things get more involved. The reasons for needing to use channel-based CSP models are equally valid in this bipartite model, but the use of these models are not quite as easy to justify since processes no longer have the "no selective input" property. Just as with the argument earlier that demonstrated that the $\mathcal{N}$ semantics of divergence is correct despite the fact that processes are not closed, we need to move outside CSP's established "comfort zone". In order for the channel-based models to be valid, we need to ensure that whenever two processes are running in parallel, one inputting and one outputting a value v on a given channel c, if the inputting process is in a state where it can accept any input on c then it can also accept v, even though there may be values that it does not accept.

This is always true of our CSP models of the π-calculus. This is because the only inputs that $\textsc{csp}2[x(y).P]\zeta\sigma$ cannot make are names that have been reserved for P to generate freshly, or has already generated but not output yet. These are different from (i) the names created by the external environment by the bipartite structure, (ii) names created by other parallel processes, because of the way we use Π_1' and Π_2', and (iii) names previously output by P, by construction.

$\textsc{csp}2[P]$ suffers from the same failures of abstraction due to exact choices of fresh names as the unified version $\textsc{csp}[P]$. We have the same two choices of how to fix these, using the same operators $\mathit{NFN}(P,\sigma)$ and $\langle\!\langle \mathit{SFN}(\kappa,\sigma)\rangle\!\rangle$. The first of these can be used without alteration, as can the second if it is only used at the outside of the semantics. We cannot incorporate this version into a denotational semantics $\mathcal{M}^2_\mu[\![P]\!]\rho\zeta\sigma$ since it requires the parameter κ. There are two choices: either to incorporate this parameter into the semantics, or to use a modified $\mathit{SFN}'(\sigma)$ that does not standardise how external names appear to the process P, something that is not necessary semantically.

It is interesting to note that one of our four CSP semantics, namely $\mathcal{M}^2_\sqcap[\![P]\!]\zeta\sigma$, is defined without any use of generalised renaming, though it does use both elaborate renamings and parallel composition with processes that limit traces.

We have already seen that the two unified semantics give the same equivalence, and the equivalence they give is independent of reasonable choices for κ and σ. A simpler arguments of the same type work for the bipartite ones semantics. Therefore we will know that all four agree if we can prove that the two nondeterministic semantics agree.

Theorem 2. *For a finite-behaviour channel-based CSP model $\mathcal{M}$ and π-calculus process P, we have*

(i) $\mathcal{M}^2_\sqcap[\![P]\!](\mathit{Names}-\sigma)\sigma = \{b \in \mathcal{M}^1_\sqcap[\![P]\!](\mathit{fn}(P))\sigma \mid \mathit{enames}(b)\cap\sigma = \emptyset\}$ *whenever* $\sigma \cap \mathit{fn}(P)=\emptyset$ *and both* σ *and* $\mathit{Names}-\sigma$ *are infinite.*

(ii) $\mathcal{M}^1_\sqcap[\![P]\!]\mathit{fn}(P)\sigma = \bigsqcap\{\mathit{MNBP}(\mathit{Names}-\sigma')\sigma' \mid \sigma' \subseteq \sigma, \mathit{Names}-\sigma'$ *is infinite for any infinite* σ *disjoint from* $\mathit{fn}(\sigma)$.

– *Consequently, the congruences induced by* $\mathcal{M}^1_\sqcap[\![\cdot]\!]$ *and* $\mathcal{M}^2_\sqcap[\![\cdot]\!]$ *are the same.*

Part (i) of this result says that behaviours b of P in a bipartite name space are just those that happen in the unified name space where the environment did not choose to inject any names that it would have been banned from doing so under bipartite rules. The fact that we are in a channel-based model is vital here.

Part (ii) essentially observes that for any finite behaviour b of the unified semantics, there is an infinite subset σ' of σ left when we remove $\mathit{enames}(b)$, and b will belong to $\mathit{MNBP}(\mathit{Names}-\sigma')\sigma'$.

It follows that the choice of which semantics to adopt is largely down to personal taste. For the author, the unified nondeterministic semantics has much to recommend it because it removes the arbitrary choices implicit in the choice of an enumeration for Names −

the semantic value generated by *NFN* is independent of which enumeration is used in constructing CSP$[\cdot]$ – and a partition of that set into process and environment names. On the other hand it seems likely that the bipartite standardised approach is best suited to most automatic verification methods.

6 Properties and future work

We have simultaneously introduced a large number of new semantics for π-calculus, simply because there is one[6] for every CSP model that does not give a representation to infinite traces. We cannot hope to explore all their properties here. In this section, therefore, we restrict ourselves to establishing a few basic properties and setting out a research programme to explore further possibilities.

Throughout this section, the $\mathcal{M}$ semantics of a π-calculus process P will be understood to mean the standardised (*SFN*) semantics calculated over the channel-based CSP model $\mathcal{M}$ over a unified name space with parameters $\mathit{fn}(P)$ and $\mathit{Names} - \mathit{fn}(P)$

6.1 Refinement and refinement checking

The most obvious consequence of having a CSP semantics is that it gives a language a natural notion of refinement. We define refinement, relative to a given model $\mathcal{M}$, by

$$P \sqsubseteq_M Q \equiv \mathcal{M}^1_\mu[\![P]\!]\kappa\sigma \sqsubseteq \mathcal{M}^1_\mu[\![Q]\!]\kappa\sigma$$

for any κ containing $\mathit{fn}(P) \cup \mathit{fn}(Q)$, which is the same as the $\mathcal{M}$-equivalence, *as processes*, of P and $\tau.P + \tau.Q$ (simply a translation of the CSP construct $P \sqcap Q$). This equational characterisation means that the definitions of refinement do not depend on which of the four options is chosen.

Our constructions, Lemma 4, and the properties of refinement in CSP imply that $\sqsubseteq_M$ has the compositionality properties one would want, namely

$$P \sqsubseteq_M Q \Rightarrow C[P] \sqsubseteq_M C[Q]$$

for any context C.

As in CSP, one process refines another if its behaviours are a subset of those of the second process. Thus one process trace refines another if all its traces are ones of the second, and so on.

In CSP models that record only finitely observable behaviours (a category that does not include divergence), there is a refinement-maximal process, always equivalent to the simply divergent term $\mu\, p.\tau.p$. There is a refinement-minimal member of the CSP model, but no π-calculus process represents it since the minimal member has the capability of using every name on the first step as an output channel. No π-calculus term can do that. It is possible to construct a minimal process in the traces model subject to knowing the set of names K initially: imagine a system constructed using the replication $!C$ and $!D$ of two sorts of cell. A C is initialised with one name x and can then endlessly either input $x(y)$ or output a fresh name $\overline{x}z$ where z is introduced by $\boldsymbol{\nu} z$ (and in either case it initialises a further C with name y or z) or initialise a D with value x.

$$C = c(x).\,\mu\, p.\boldsymbol{\nu} z.(x(y).\overline{c}y.p + \overline{x}z.\overline{c}y.p + \overline{d}x.p)$$

[6] We say "one" here, because the four options given for presenting it are almost certainly congruent to each other.

D is initialised with two names and then endlessly outputs one over the other.

$$D = d(x).d(y).\,\mu\, p.\overline{x}y.p$$

The process is then defined $\boldsymbol{\nu} c \boldsymbol{\nu} d\,!C\ |!D \mid C'[k_1/x] \mid \ldots \mid C'[k_n/x]$ where C' is an initialised copy of C.

The $!C = \mu\, p.C \mid \tau.p$ construction could not be used in the same way in building refinement-minimal elements of other finite-behaviour models such as stable failures $\mathcal{F}$ because it is never itself stable. It is, nevertheless, possible to adapt the above construction to build a refinement-minimal element for $\mathcal{F}$ for a given initial knowledge K. This is left as an exercise to the ingenious reader!

In FDR, the main mode of verifying processes is to show that they refine other CSP processes representing their specifications. Since our constructions have, in effect, embedded π-calculus within CSP, we have considerable freedom in how to formulate refinement checks using a mixture of the two languages.[7] A reasonably general model is provided by checks of the form

$$\mathit{Spec} \sqsubseteq_M C[\mathit{Imp}]$$

where *Spec* is either a CSP or π-calculus process and C is a possibly null CSP context.

So, for example deadlock freedom is equivalent to the refinement check

$$\mu\, p.\tau.p \sqsubseteq_F \mathit{Imp} \setminus \mathit{Events}$$

and we can use the *lazy abstraction* construction from CSP, as in the following *fault tolerance* check from Chapter 12 of [16] that goes a little beyond the model above:

$$\mathit{Imp} \underset{E}{\parallel} \mathit{STOP} \sqsubseteq_F (\mathit{Imp} \underset{E}{\parallel} \mathit{CHAOS}(E)) \setminus E$$

where E is a set of events that trigger erroneous behaviour within *Imp*. This specification says that whatever externally visible behaviour appears with these error events allowed also happens when they are banned.

6.2 Prospects for using FDR

Having formulated specifications as refinement checks, it is interesting to ask whether they can be run on FDR, since this might provide a powerful additional tool to apply to systems designed in π-calculus.

FDR achieves its considerable speed by concentrating on finite-state systems, allowing most checks to be performed in time linear (or sub-linear if compressions are used) in the number of states of the implementation. It concentrates on what it the central case of CSP, namely of a number of finite-state processes connected by a static harness built of parallel, hiding and renaming operators.

This is challenged by two aspects of π-calculus. Firstly, π-calculus assumes that every $x(y)$ input communication offers an infinite range of inputs. Secondly, π-calculus networks are frequently dynamic and potentially unbounded in size, as seen in the construction above that built the trace-refinement-minimal process. This takes them outside FDR's "comfort zone" and probably out of its range altogether.

[7] In such usage it will be necessary to ensure that any CSP used respects the no-selective-input regime required to make channel-based models works.

It therefore seems sensible to start by looking at π-calculus networks that create static networks and which generate only finitely new names using $\boldsymbol{\nu} z\, P$ constructs.

We therefore consider networks which take the form

$$\boldsymbol{\nu} x_1\, \boldsymbol{\nu} x_2 \ldots \boldsymbol{\nu} x_n\, (P_1 \mid P_2 \mid \ldots \mid P_m)$$

where the P_i make no use of the operators $\mid$ and $\boldsymbol{\nu} x$ and are therefore finite-state except for the precise names they contain.

The author conjectures that the majority of trace refinement checks involving such processes can be performed finitely in the following way, thanks to Lazić's theory of data independence in CSP [8].

- Use the bipartite approach to semantics.
- Use a type of $n + k + 2$ names, of which n are tied to the identifiers $x_1, \ldots, x_n$, k are the externally visible names in the system, and the final two are "extras". The reason there are just 2 extra names is that every π-calculus term, with the exception of the inequality constraints for fresh variables, satisfies the condition **PosConjEqT** from [8], which states that no two members of the data independent type (here *Names*) are every compared for equality except in such a way that the consequences of equality are always trace refined by the consequences of inequality. In π-calculus, both (obviously) equality guards $[x = y]$ and communication in $P \mid Q$ represent equality tests of this sort. The first use of any extra name must be as an input along a pre-existing channel. Lazić's results show that in similar circumstances, it usually suffices to consider a check in which the data independent type is of size one or two greater than the number of distinct constant values. This represents a *threshold* value for the refinement check.
  Thus initially, and until the process has extruded one or more of the names x_i, inputs are restricted to the other $k + 2$ names L.
- Regarding the identifiers x_i as constants, replace the system with $(P_1 \mid \ldots \mid P_m) \underset{\Sigma_\pi}{\|} BOF(L)$.
- This process may well not be symmetric in the names x_i, but if it is refinement checked against a process that is symmetric in them (for example by not referring to them at all, as will presumably be the case when the system does not extrude names) it should be equivalent to check that the unsymmetric version refines the specification.

Data independence will, in fact, be applicable to all π-calculus descriptions of systems: it is impossible to create π-calculus processes that are not data independent, at least using the syntax we have adopted for it.

Another class of system that uses an unbounded collection of fresh values has already been widely studied in CSP, namely cryptographic protocols where the values are keys, nonces etc. (This connection has already been exploited in the Spi Calculus [1].) Several related methods for turning naturally infinite-state refinement checks involving these into finite-state ones have been developed for these protocols[21, 7]. These methods use a combination of recycling values that no longer have any meaning to the system and making room fro this by identifying "out of date" values.

It would be interesting to see whether these same techniques can be employed usefully for π-calculus systems that generate an infinite number of fresh names in their lifetimes but only have a finite number meaningful at any one time.

Using the present version of FDR it is not possible to analyse networks that grow unboundedly. It should, however, be possible to analyse ones that are dynamic but only have a bounded number of active (i.e. non-0 processes in π-calculus terms) at any one time and only have a finite number of patterns that these processes follow. We would need to create a CSP process definition *Flex* that could be initialised to behave like one of these patterns and returns to the state *Flex* once the first pattern has finished. When a process executes a parallel command $P \mid Q$ it would implement one of these (say P) itself and initialise one of the available *Flex*es to behave like Q.

6.3 Full abstraction

All the well-known CSP models are fully abstract with respect to one or more simple tests that can be made of processes in the sense that two processes P and Q are identified in the model if and only if, for all CSP contexts $C[\cdot]$, $C[P]$ passes each of the tests if and only if $C[Q]$ does. Thus, for example, the finite traces model $\mathcal{T}$ is fully abstract with respect to the test "P has the trace $\langle a \rangle$" for an arbitrarily chosen visible event a, and the failures divergences model $\mathcal{N}$ is fully abstract with respect to the test "neither deadlocks nor divergences on $\langle\rangle$".

It is clear that, when a given CSP model $\mathcal{M}$ has such a property, then two π-calculus terms P and Q are identified in it if and only if all *CSP* contexts applied to them give the same results for the tests. It is interesting to ask whether the same will hold for *π-calculus* contexts. The author believes that most of these results will indeed carry across in this way, but resolving these questions is beyond the scope of this paper.

6.4 Comparison

Equivalences for π-calculus terms have previously been given in terms of bisimulation relations of different sorts. Over other process calculi it is true that bisimulations relations generally give *finer* equivalences than CSP-style models, except that some bisimulations (e.g. weak) do not make the same distinctions based on the potential to diverge that some CSP models do. The author expects that the same will be true for π-calculus.

It seems to the author that the semantics of π-calculus in CSP models have the merit of simplicity, in the sense that each recorded behaviour only gives a single value to each name, whichever of our four options is picked. These models seem no less natural for π-calculus than they do for CSP. Indeed, given that in a unified name space it is inevitable that any branching-lime model for π-calculus there will be recorded behaviours on which the name declared in $\boldsymbol{\nu} z\, P$ will appear with two different values, the use of linear observations seems *particularly* appropriate here.

7 Conclusions

We have illustrated the enormous expressive power of Hoare's CSP by giving a number of semantics to π-calculus. The author hopes that in addition to providing this demonstration of the power of CSP this paper will also be the key to new understandings and process analysis methods for π-calculus, and that for some audiences it might provide a relatively comprehensible way of explaining the semantics of that notation. In particular it will be interesting to see if the existence of compositional theories of refinement for π-calculus will have any applications.

In addition to the topics for further work highlighted in the previous section, there is a further open end to be explored, namely the topic of models with infinite traces. The author suspects that it may be possible to modify our CSP translation so that this works with a countably infinite name space using the bipartite approach, but not using the unified approach to fresh names.

While this may or may not be apparent to the reader, the author discovered on numerous occasions that the semantic decisions made in the design of π-calculus were absolutely crucial to the creation of a reasonably elegant semantics for it in CSP. A prime example of this is the rule that no fresh name can be used as a channel until it has been passed along another channel is necessary for ensuring that the first time a name appears in a behaviour in a channel-based model is as the "data" field of an actually communicated event. This is key to a number of things working properly in the CSP semantics. Thus, at least to the author, this work demonstrated not only the power of CSP but also the great elegance of the π-calculus.

References

1. M. Abadi and A.D. Gordon, *A calculus for cryptographic protocols: the Spi calculus*, Proceedings of the 4th ACM conference on Computer and communications security, 1997.
2. S.D. Brookes, *A model for Communicating Sequential Processes*, Oxford University DPhil thesis, 1983.
3. S.D. Brookes, C.A.R. Hoare and A.W. Roscoe, *A theory of communicating Sequential Processes*, Appeared as monograph PRG-16, 1981
   `http://web.comlab.ox.ac.uk/people/Bill.Roscoe/publications/1.pdf`
   and extended in JACM **31** pp 560-599, 1984. 1
4. S.D. Brookes, A.W. Roscoe and D.J. Walker, *An operational semantics for CSP*, Oxford University Technical Report, 1986. 25
5. He Jifeng and C.A.R. Hoare, *CCS is a retract of CCS*, Unifying Theories of Programming symposium, Springer LNCS 4010, 2006.
6. C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
7. Eldar Kleiner, *A web services security study using Casper and FDR*, Oxford University DPhil thesis, 2008.
8. R.S. Lazić, *A semantic study of data-independence with applications to the mechanical verification of concurrent systems*, Oxford University D.Phil thesis, 1998.
9. R. Milner, *A Calculus of Communicating Systems,* LNCS 92, 1980
10. R. Milner, *Communication and concurrency*, Prentice-Hall, 1989
11. R. Milner, *Communicating and mobile systems: the π-calculus*, CUP, 1999
12. R Milner, J Parrow and D Walker, *A calculus of mobile systems, Parts I/II*, Information and Computation, 1992
13. A.W. Roscoe, *A mathematical theory of Communicating Sequential Processes*, Oxford University DPhil thesis, 1982. 2
14. A.W. Roscoe, *Denotational semantics for occam*, Proceedings of the 1984 Pittsburgh Seminar on Concurrency, Springer LNCS 197. 10
15. A.W. Roscoe, *model-checking CSP*, in *A Classical Mind, essays in honour of C.A.R. Hoare*, Prentice-Hall 1994. 50
16. A.W. Roscoe, *The theory and practice of concurrency*, Prentice-Hall International, 1998. Updated version available at 68.
17. A.W. Roscoe, *Seeing beyond divergence*, Communicating Sequential Processes, the first 25 years, Springer LNCS 3525, 2005. 95
18. A.W. Roscoe, *Revivals, stuckness and the hierarchy of CSP models*, JLaP **78**, 3, pp163-190, 2009. 105
19. A.W. Roscoe, *The three platonic models of divergence-strict CSP* Proceedings of ICTAC 2008. 121
20. A.W. Roscoe *On the expressiveness of CSP*, Submitted for publication. 125
21. A.W. Roscoe and P.J. Broadfoot, *Proving security protocols with model checkers by data independence techniques*, Journal of Computer Security **7**, pp147-190, 1999. 70
22. A. W. Roscoe, P.H.B. Gardiner, M.H. Goldsmith, J.R. Hulance, D.M.Jackson and J.B. Scattergood, *Hierarchical compression for model-checking CSP, or How to check $10\hat{2}0$ dining philosophers for deadlock*, In Proceedings of TACAS 1995 LNCS 1019. 59
23. D. Sangiorgi and D. Walker *The π-calculus: A theory of mobile processes*, CUP, 2001.

24. J.E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.
25. R.J. van Glabbeek, *On cool congruence formats for weak bisimulation*, Proceedings of ICTAC 2005, Springer LNCS 3722.
26. P.H. Welch and F.R.M. Barnes, *A CSP model for mobile processes*, Proc CPA 2008 (IOS Press, 2008).

All the works listed above whose authors include A.W. Roscoe can be found at `http:web.comlab.ox.ac.uk/people/publications/personal/Bill.Roscoe.html`. The numbers `1` etc refer to their respective numbers on that web page.