Available online at www.sciencedirect.com
ScienceDirect
The Journal of Logic and Algebraic Programming 75 (2008) 139–165
THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

CCS with priority guards ☆

Iain Phillips *

Department of Computing, Imperial College London, London SW7 2AZ, England, United Kingdom

Available online 16 June 2007

Abstract

It has long been recognised that standard process algebra has difficulty dealing with actions of different priority, such as for instance an interrupt action of high priority. Various solutions have been proposed. We introduce a new approach, involving the addition of "priority guards" to Milner's process calculus CCS. In our approach, priority is *unstratified*, meaning that actions are not assigned fixed levels, so that the same action can have different priority depending where it appears in a program. Unlike in other unstratified accounts of priority in CCS (such as that of Camilleri and Winskel), we treat inputs and outputs symmetrically. We introduce the new calculus, give examples, develop its theory (including bisimulation and equational laws), and compare it with existing approaches. We use leader election problems to show that priority adds expressiveness to both CCS and the π-calculus.

Keywords: CCS; Priority; Expressiveness; π-calculus

1. Introduction

It has long been recognised that standard process algebra [17,13,2] has difficulty dealing with actions of different priority, such as for instance an interrupt action of high priority. Various authors have suggested how to add priority to process languages such as ACP [1,14], CCS [7,5] and CSP [11,10]. We introduce a new approach, involving the addition of "priority guards" to the summation operator of Milner's process calculus CCS. In our approach, priority is *unstratified*, meaning that actions are not assigned fixed levels, so that the same action can have different priority depending where it appears in a program. We shall see that existing accounts of priority in CCS are either stratified [7], or else they impose a distinction between outputs and inputs, whereby prioritised choice is only made on inputs [5,8]. Such a distinction is somewhat restrictive, though one can probably work around the issue in actual examples. Although one can see a technical need for the distinction in the case of [8], it is hard to see a fundamental reason why it should be needed. It also goes against the spirit of CCS, where inputs and outputs are treated symmetrically. We contend that it is unnecessary.

We introduce the new calculus, give examples, develop its theory (including bisimulation and equational laws), and compare it with existing approaches. We use leader election problems to show that priority adds expressiveness to both CCS and the π-calculus.

☆ A shorter version of this paper appeared in Concur 2001 [24].

* Tel.: +44 (0)20 7594 8265; fax: +44 (0)20 7581 8024.
E-mail address: iccp@doc.ic.ac.uk

1567-8326/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2007.06.005

We start with the idea of priority. Recall that, in CCS, communication involves exactly two parties, which experience the interaction simultaneously (so-called "hand-shaking"). We assume some familiarity with CCS notation [17,19]. Consider the CCS process $a + b$. The actions a and b have equal status. Which of them engages in communication depends on whether the environment is offering the complementary actions $\overline{a}$ or $\overline{b}$. By "environment" we mean whatever processes may be placed in parallel with $a + b$. We would like some means to favour a over b, say, so that if the environment offers both, then only a can happen. This would be useful if, for instance, a was an interrupt action. We need something more sophisticated than simply removing b altogether, since, if a cannot communicate, it should not stop b from doing so. This brief analysis points to two features of priority: (1) Priority removes ("preempts") certain possibilities that would have existed without priority. Thus if a can communicate then b is preempted. (2) Reasoning about priority in CCS has to be parametrised by the environment.

We now explain the basic idea of priority guards. Let P be a process, let a be an action, and let U be some set of actions. Then we can form a new process $U:a.P$, which behaves like $a.P$, except that the initial action a can only be performed if the environment does not offer any action in $\overline{U}$, the CCS "complement" of U. We call U a *priority guard* in $U:a.P$. All actions in U have priority over a at this point in the computation. We call our calculus CPG (for CCS with Priority Guards).

As a simple example, if we have a CCS process $a.P + b.Q$ and we wish to give a priority over b in the choice, we add a priority guard to get $a.P + a:b.Q$ (we omit the set braces around a). Priority is specific to this choice, since the guard affects only the initial b, and not any further occurrences of b there may be in Q.

Let us see how this example is handled in two existing approaches to priority. Cleaveland and Hennessy [7] add new higher priority actions to CCS. They would write our example as $\underline{a}.P + b.Q$ (high priority actions are underlined). In their stratified calculus, actions have fixed priority levels, and only actions at the same priority level can communicate. In this paper we are interested in an unstratified approach, and so our starting point of reference is Camilleri and Winskel's priority choice operator [5]. In their notation the example becomes $a.P \mathbin{+\!\!\rangle} b.Q$. They make the priority of a over b specific to the particular choice, so that b might have priority over a elsewhere in the same program. We shall compare our approach in detail with these two existing ones in Section 2.

We have said that a priority guard prevents an action going ahead if a higher-priority action is "offered" by the environment. This notion of "offer" needs to be defined before we know what actions can actually take place. In CPG we take a very simple view, which is that any immediately available action is offered, regardless of whether it can actually occur. So an action can be offered, even though its partner co-action is unavailable, and even if the action is preempted by other higher-priority actions. In this respect CPG differs radically from other treatments of priority, which adopt what we call the "Principle of Limited Preemption" (Section 2.3), meaning that an action which is itself preempted cannot preempt other actions. In this paper we show that CPG-style preemption yields a successful and tractable treatment of priority. Therefore there are apparently at least two different meanings of "priority" in CCS. It remains an open question as to whether either meaning can be shown to be consistent with an independent model.

Since offers can only be made by immediately available actions, even silent (τ) actions can delay an offer from being made. So the process $\tau.a$ is different from a, in that the former does not offer a immediately. This difference is apparent when we place the process $\overline{a}:b$ in parallel. Indeed, $a \mid \overline{a}:b$ cannot perform b (since, as we shall see, b is preempted by the offer of a), whereas $\tau.a \mid \overline{a}:b$ can perform b initially, as a is not offered until τ has occurred.

In standard CCS, a process such as $a \mid \overline{a}$ can perform a communication, producing a τ action. In CPG, such communications are no longer autonomous, but are conditional on the environment. Thus the process $b:a \mid \overline{a}$ will only perform τ if the environment is not offering $\overline{b}$. Two CPG processes can only communicate if for each process, the environment (which includes the other process) is not offering any action corresponding to a priority guard of the process.

We start the technical development by defining the language of CPG and the reaction and labelled transition relations. We then proceed to define appropriate notions of strong and weak (i.e. taking into account silent actions) bisimulation. The associated equivalences are shown to be congruences (Theorems 6.3 and 7.4). The intuition behind our notion of bisimulation is that for processes to be equivalent they must make the same offers, and for a process Q to simulate a process P, Q must be able to do whatever P can, though possibly constrained by fewer or smaller priority guards. For instance, we would expect the processes $a + u:a$ and a to be equivalent, since the priority guarded $u:a$ is simulated by a. We develop simple equational laws which are complete for finite processes in both the strong and weak cases (Theorems 6.14 and 7.10). Even the weak laws are considerably simpler than the ones Camilleri and Winskel give for the strong case.

We take care throughout that the new language is *conservative* over CCS. As far as transitions are concerned, this requires that no new transitions are created for standard CCS processes. But we also ensure that both the strong and the weak congruences are conservative over the standard notions. To this end the language incorporates *priority actions*, which are the only ones which have a preemptive effect. This is not a stratification as such, since priority actions are only prioritised where they appear in priority guards. If conservation over CCS was not an issue, they could be dispensed with.

In applications we envisage that standard CCS reasoning can be used most of the time, and CPG reasoning will only be needed in those subsystems which use priority. As well as conservation, we rely here on a key feature of CPG, which is that priority can be *localised* using the restriction operator; in fact, a process where all priority names are restricted is equivalent to a CCS process (Proposition 6.7). Priorities which are only relevant to subsystems can be encapsulated locally, and as a result can be ignored outside their own subsystems. We regard this as an important feature when programming larger systems, where different priority frameworks may be in use in different subsystems. Thus both conservation and localisability are useful in building systems and reasoning about them in a modular fashion.

We now give an example to illustrate the handling of hidden actions and the scoping of priority. We wish to program a simple interrupt. Let P be a system which consists of two processes A, B in parallel which perform actions a, b respectively, while communicating internally (on channel mid) to keep in step with each other. P also has an interrupt process I which shuts down A and B when the interrupt signal int is received.

$$P \stackrel{\text{def}}{=} \mathsf{new}\ \mathsf{mid}, \mathsf{int}_A, \mathsf{int}_B\ (A \mid B \mid I) \qquad A \stackrel{\text{def}}{=} \mathsf{int}_A : a.\overline{\mathsf{mid}}.A + \mathsf{int}_A$$
$$I \stackrel{\text{def}}{=} \mathsf{int}.(\overline{\mathsf{int}}_A \mid \overline{\mathsf{int}}_B) \qquad B \stackrel{\text{def}}{=} \mathsf{int}_B : b.\mathsf{mid}.B + \mathsf{int}_B$$

Without the priority guards in A and B, P could receive an int and yet A and B could continue with a and b. Actions int_A, int_B have priority over a, b, respectively. This only applies within the scope of the restriction. We can apply the usual techniques of CCS (including removing τ actions) and get

$$P = a.P_1 + b.P_2 + \mathsf{int} \quad P_1 = b.P + \mathsf{int} \quad P_2 = a.P + \mathsf{int}$$

which is what we wanted (equality here means weak bisimulation equivalence, which is a congruence).

Notice that in the system as a whole, once the high-priority interrupt action is restricted, we have regained a standard CCS process without priority, illustrating the fact that priority can be encapsulated locally, as discussed above.

It is also worth noting that if we merely wanted to interrupt A and B, we could have simply defined $A' \stackrel{\text{def}}{=} \mathsf{int}_A : a.\overline{\mathsf{mid}}.A'$, $B' \stackrel{\text{def}}{=} \mathsf{int}_B : b.\mathsf{mid}.B'$. As soon as int occurs, I offers $\overline{\mathsf{int}}_A$, $\overline{\mathsf{int}}_B$, so that A' and B' cannot perform a, b respectively according to our rules. This shows that priority guards do not require summation to have an effect. The summands int_A, int_B in A, B give us a way of gracefully recovering from the interrupt so that further computation could take place (if programmed after the int_A, int_B guards).

We shall return to this example to give more precise details in Section 8.

A striking by-product of adding priority guards to CCS is that we can encode mixed input and output guarded summation using priority guards and restricted parallel composition, provided the summands cannot react with each other (Proposition 6.15). As a simple example, $a.P + \overline{b}.Q$ can be encoded with priority guards as $\mathsf{new}\ c\ (c : a.(P \mid \overline{c}) \mid c : \overline{b}.(Q \mid \overline{c}))$ (where c is a fresh action). This expresses in a natural way the preemptive nature of $+$, whereby pursuing one option precludes the others. The same effect can be achieved in Camilleri and Winskel's calculus (but only for input guards): $a.P + b.Q$ can be encoded as

$$\mathsf{new}\ c\ ((c +\!\rangle\ a.(P \mid \overline{c})) \mid (c +\!\rangle\ b.(Q \mid \overline{c})))$$

but of course here we are simply exchanging one form of choice for another.

This encoding of summation is similar to Nestmann and Pierce's encoding of input-guarded summation in the asynchronous π-calculus [22], where the idea is that the summands compete in parallel to be the first to successfully interrogate a lock. The Nestmann–Pierce encoding would also work without name passing, but it does need asynchronous communication, whereas of course communication in CCS is synchronous. In our case, the added power of priorities means that we are able to handle certain mixed choices, and not just input-guarded choice.

We conclude the paper by studying the expressiveness of CPG via leader election problems, building on the work of Palamidessi [23] and Ene and Muntean [9]. We show that the leader election problem can be solved uniformly for any size of network in CPG (Theorem 9.10). Here "uniformly" means that the individual processes have a definition which is independent of the size of the network. This illustrates the fact that in CPG high priority actions can have a

"broadcast" effect over their environment, in that a single process can simultaneously interrupt several other processes. By contrast, CCS and π-calculus communication are always one–one. We then use Theorem 9.10 to show that CPG cannot be encoded in the π-calculus [20] under certain conditions (Theorem 9.16). Since CCS can be seen as a subcalculus of the π-calculus, this shows that adding priority guards produces an increase in expressive power in the case of both CCS and the π-calculus. Of course, adding expressive power has been the very intention of all proposals for introducing priority constructs, but, as far as we are aware, it has not been proved previously that any notion of priority adds expressiveness to a process calculus.

Palamidessi showed that the π-calculus is more expressive than CCS by showing that the leader election problem can be solved for a ring of four processes in the π-calculus, but not in CCS. We show that in fact CPG is powerful enough to solve the problem for a ring of size four (Proposition 9.23), but not of size any composite (non-prime) number larger than four (Theorem 9.24). Noting that Palamidessi's work easily leads to a solution in the π-calculus for rings of any size, we deduce that the π-calculus is not encodable in CPG (Theorem 9.27) under certain conditions.

The rest of the paper is organised as follows. First we compare our approach with related work (Section 2). Next we define the language of processes (Section 3). Then we look at reactions (Section 4) and labelled transitions (Section 5). We then look at bisimulation and equational theories for both the strong (Section 6) and weak cases (Section 7). We then return to our interrupt example (Section 8), and look at the extra expressiveness afforded by priority guards (Section 9). The paper is completed with some brief conclusions.

2. Comparison with related work

We refer the reader to [8] for discussion of the many approaches taken by other authors to priority. Here we restrict ourselves to comparison of our work with that of Camilleri and Winskel [5] (referred to as CW for short) and Cleaveland, Lüttgen and Natarajan [8] (CLN for short).

2.1. Camilleri and Winskel (CW)

As we have seen, CW's CCS with a prioritised choice operator $P \mathbin{+\!\!\rangle} Q$ allows priority to be decided in a way which is specific to each choice in a system. The idea of a priority choice between processes is interesting and natural. The authors present an operational semantics via a labelled transition relation, and define a bisimulation-based equivalence. They also give an axiomatisation of this equivalence which is complete for finite processes (i.e. those not using recursion). They do not show how to hide the τ-actions resulting from communications (though this is treated in [15]).

As we saw in Section 1, reasoning about priority has to be parametrised on the environment. The CW transition relation is parametrised on a set of output actions R. Thus $\vdash_R P \stackrel{\alpha}{\to} P'$ means that, in an environment which is ready to perform precisely the actions R, the process P can perform an action α to become P'. For example, $\vdash_R a \mathbin{+\!\!\rangle} b \stackrel{a}{\to} \mathbf{0}$ (any R), while $\vdash_R a \mathbin{+\!\!\rangle} b \stackrel{b}{\to} \mathbf{0}$ provided $\overline{a} \notin R$.

We have borrowed the idea of parametrisation on the environment for our labelled transition system for CPG. For us $P \stackrel{\alpha}{\to}_U P'$ means that, in an environment which offers no action in the set $\overline{U}$, process P can perform α to become P'. Our most basic rule is essentially $U : a.P \stackrel{a}{\to}_U P$, provided $a \notin U$.

Note that the CW syntax shows the environment only in the prioritised choice $a.P \mathbin{+\!\!\rangle} b.Q$, and does this implicitly, in that $b.Q$'s "environment" is $a.P$, while $a.P$ says nothing about the actual environment. In CPG the environment is represented in the syntax directly.

There is a difference in expressiveness between CPG and CW's calculus, in that the latter cannot express cycles of priority, whereas we can in CPG. CW consider the paradoxical example $\mathsf{new}\ a, b\ ((a \mathbin{+\!\!\rangle} \overline{b}) \mid (b \mathbin{+\!\!\rangle} \overline{a}))$. The problem is that there is a circularity, with a having priority over b, as well as vice versa. Can the system act? They decide to sidestep this question by breaking the symmetry in CCS between inputs and outputs, and only allowing prioritised choice on input actions. We feel that this complicates the syntax and operational semantics, and should not be necessary. There seems to be no essential reason for CW not to allow the system with circular priorities, since their environmental transition relation should be able to handle it. In our approach the example is admitted, and results in a deadlock, which would seem to be in keeping with CW's approach. We consider this example again in Section 5.

Another reason why CW disallow priority choice on output actions is to assist in obtaining the normal form they use for proving the completeness of their equational laws for finite processes. However this normal form is still quite complicated (consisting of a sum of priority sums of sums). In our calculus CPG we have only one form of choice, and so completeness is technically simpler.

There seems to be no telling reason for priority choice to be attached to input guards rather than output guards. Perhaps the intuition is that it should be easier to implement, much in the way that input choice is considered easier to implement than output choice in the π-calculus (being encodable in the asynchronous π-calculus [22]). In Prasad's calculus CBS (with broadcast rather than hand-shaking communication) priorities are similarly localised to one of the partners in a communication [27]. But in this case they are attached to output actions, which makes sense, since they are the autonomous ones. Preempting an input would make no sense, since the absence of an input does not inhibit an output.

In CCS, only silent (τ) actions are autonomous. In CPG, we shall allow them to be prefaced by priority guards, and thereby to be preempted. But of course τ actions are meant to represent a handshake between an action and a co-action. Unlike in CBS, both sides of the handshake are needed, and we allow priority guards to be attached to either side. In order for a handshake to go ahead, neither side must be preempted by its environment.

2.2. Cleaveland, Lüttgen and Natarajan (CLN)

In CLN's basic approach [8], which is derived from earlier work of Cleaveland and Hennessy [7] (referred to as CH for short), actions have priority levels. Mostly CLN consider just two levels—ordinary actions and higher priority, underlined actions. Only actions at the same level of priority can communicate, which is really quite restrictive when one considers that two actions which are intended to communicate may have quite different priorities within their respective subsystems. Silent actions resulting from communication have preemptive power over all actions of lower priority. The authors present both strong and weak bisimulation-based equivalences (drawing on [21]), and axiomatise these for finite processes.

In our unstratified calculus CPG, by contrast, actions do not have priority levels—each priority guard operates independently, in the spirit of [5].

Even disregarding the issue of priority levels, there is a difference between preemption in [7,8] and in CPG, since in CPG preemption is done entirely by the environment. Consider the process $a + \underline{\tau}$. According to CH, this process cannot perform a, since it is preempted by $\underline{\tau}$ (similarly, for CW the process $\tau +\!\rangle\, a$ is equivalent to τ). However if we translate $a + \underline{\tau}$ into CPG as $U : a + \tau$ (where the set of actions U, with $a \notin U$, is chosen to be as large as necessary), we find that by contrast $U : a + \tau \xrightarrow{a}_U \mathbf{0}$. So the a is not preempted, but only downgraded (performance of a depends on the environment not offering any action in $\overline{U}$).

Another example illustrates the opposite effect. For CH, $a \mid \underline{b} \xrightarrow{a} \underline{b}$, but the translation $U : a \mid \underline{b}$ (with $\overline{\underline{b}} \in U$) cannot perform a at all, since the environment (in the form of $\underline{b}$) is offering an action which preempts a. Here we see that, in CPG, a high priority action can preempt a low priority action in a parallel composition, even without being able to engage in a reaction.

This difference in the handling of preemption means that there is no obvious translation of CPG into the framework of CLN, or vice versa.

We referred in Section 1 to the desirability of encapsulating priorities locally. This encapsulation is present in CW's calculus (and in our own), but not in CH's, since they treat a high priority $\underline{\tau}$ differently from a standard τ. However, the development in [8] goes beyond the basic CH calculus to consider distributed priorities, where preemption is decided locally rather than globally.

CLN motivate this by the example of an application which fetches data from two memory benches alternately. In CCS this can be modelled as

$$\mathsf{Appl} \stackrel{\text{def}}{=} \overline{\mathsf{fetch}_1}.\overline{\mathsf{fetch}_2}.\mathsf{Appl}$$

These benches are also connected to a direct-memory-access (DMA) controller. This DMA access should have lower priority than the fetch access by the application. However a straightforward assignment of high priority to application access and low priority to DMA access fails, since one or other of the fetches is always enabled, so that DMA access never takes place.

Their example can be encompassed easily in our unstratified approach. Define

$$\mathsf{Bench}_i \stackrel{\text{def}}{=} \mathsf{fetch}_i.\mathsf{Bench}_i + \mathsf{fetch}_i : \mathsf{dma}.\mathsf{Bench}_i \quad (i = 1, 2)$$
$$\mathsf{Sys} \stackrel{\text{def}}{=} \mathsf{new}\ \mathsf{fetch}_1, \mathsf{fetch}_2\ (\mathsf{Appl} \mid \mathsf{Bench}_1 \mid \mathsf{Bench}_2)$$

Then Sys has the desired behaviour, since one or other dma action can always take place. In Section 8 we show that $\mathsf{Sys} = P$, where $P \stackrel{\text{def}}{=} \mathsf{dma}.P$.

The next step CLN take is to consider extending the distributed priority calculus to allow communication between actions at different levels. The authors identify a problem with associativity of parallel composition. Consider the system

$$(a + \underline{b}) \mid (\overline{b} + \underline{c}) \mid \overline{c}$$

where communication is allowed between complementary actions at different levels. If this associates to the left, then a is preempted by b; however if it associates to the right then b is preempted by c, and so a is not preempted. A similar problem is encountered when extending the distributed calculus to allow more than two levels. CLN's proposed solution is to follow CW by only allowing priorities to be resolved between *input* actions, while treating all output actions as having equal priority. We have already mentioned our reservations about this. Nevertheless the distinction between inputs and outputs gives a workable "mixed-level" calculus (distributed, multi-level, with communication between different levels). It is particularly nice that CLN show that the CW calculus can be translated faithfully and naturally into this mixed-level calculus. This shows that the underlying model of preemption is the same in both cases, and, apparently, different from that of CPG.

It is striking that both CW and the mixed-level calculus of CLN adopt the same syntactic restriction on inputs and outputs, and also that only *strong* equivalence (τ actions not hidden) is presented for the mixed-level calculus. We shall present a weak equivalence for CPG.

2.3. The principle of limited preemption

It can plausibly be argued that an action which is itself preempted should not be allowed to preempt another action. Let us call this the principle of limited preemption (PLP):

- A disabled process (or action) cannot perform any preemption.

Disabling could be either through lack of a partner process/action, or through preemption by some higher priority process/action.

In the case of the example from Subsection 2.1 with circular priorities, both actions (a and b) are preempted, and therefore cannot perform any preemption. So, according to PLP, both a and b can react. Clearly CPG does not conform to the PLP, since it allows the environment to preempt even if it is itself preempted, so that the circular example deadlocks. Both CW and CLN do conform to PLP; however they rule out the circular example, which is a particularly strong test case for PLP.

Returning to the example $(a + \underline{b}) \mid (\overline{b} + \underline{c}) \mid \overline{c}$ of Subsection 2.2, we see that, according to PLP, a should not be preempted, since it is preempted by b, which is itself preempted. In CPG, a is preempted. Again this example is ruled out by CW and CLN.

The example can be generalised to a chain of preemption such as the CPG process

$$a_1 \mid (\overline{a}_1 + a_1 : a_2) \mid (\overline{a}_2 + a_2 : a_3) \mid \cdots \mid (\overline{a}_{n-1} + a_{n-1} : a_n) \mid \overline{a}_n$$

Here if we follow PLP we find that $a_2, a_4, \ldots$ are preempted (by $a_1, a_3, \ldots$), while $a_1, a_3, \ldots$ can react. Such calculations can become intricate. In CPG, only a_1 can react. In designing CPG, we wished to have the simplest possible notion of preemption.

Unlike CW and CLN, we wish to allow such distributed chains of priorities, partly to avoid arbitrary syntactic complications, and partly to see whether it is possible to allow them. We contend that no harm comes of violating PLP, and considerable simplification in the semantics of preemption is achieved. It may well be possible to include the problematic examples and conform to PLP, but it would seem to lead to complications.

3. The language CPG

We shall denote our augmentation of CCS with priority guards by *CPG* (CCS with Priority Guards). First we define the actions of CPG. In standard CCS [19, Part I] there is a set of *names* $\mathcal{N}$ and a disjoint set of *co-names* $\overline{\mathcal{N}} = \{\overline{a} : a \in \mathcal{N}\}$, together with a single silent action τ. To these standard actions we shall add a new disjoint set of *priority names* $\mathcal{U}$ and a dual set $\overline{\mathcal{U}}$. These are the actions which can be used in priority guards; they can also be used in the standard way. They need to be kept separate from standard actions, since we have to be careful with them in reasoning compositionally about processes.

To see why we take this approach, consider the law $P = \tau.P$, which is valid for CCS processes.[1] In CPG, if a can be a priority guard then $a \neq \tau.a$, since, as we saw in Section 1, the two sides behave differently in the context $\cdot \mid \overline{a}:b$. However if we know that a is a standard name then we do have $a = \tau.a$. So we can retain CCS reasoning when processes only involve standard names.

We define $\mathsf{Std} = \mathcal{N} \cup \overline{\mathcal{N}}$ (the standard visible actions), $\mathsf{Pri} = \mathcal{U} \cup \overline{\mathcal{U}}$ (the priority actions), $\mathsf{Vis} = \mathsf{Std} \cup \mathsf{Pri}$ (the visible actions), and $\mathsf{Act} = \mathsf{Vis} \cup \{\tau\}$ (all actions). We let $a, b, \ldots$ range over $\mathcal{N} \cup \mathcal{U}$; $u, v, \ldots$ over Pri; $\lambda, \ldots$ over Vis; and $\alpha, \beta, \ldots$ over Act. Also $S, T, \ldots$ range over finite subsets of Vis, and $U, V, \ldots$ over finite subsets of Pri. If $S \subseteq \mathsf{Vis}$, let $\overline{S}$ denote $\{\overline{a} : a \in S\}$, where if $\overline{a} \in \overline{\mathcal{N}} \cup \overline{\mathcal{U}}$ then $\overline{\overline{a}} = a$. For $\lambda \in \mathsf{Vis}$ we define $\mathsf{name}(\lambda)$ to be λ if $\lambda \in \mathcal{N} \cup \mathcal{U}$ and $\overline{\lambda}$ if $\lambda \in \overline{\mathcal{N}} \cup \overline{\mathcal{U}}$.

Now we define processes:

Definition 3.1 (*cf.* [19, *Definition* 4.1]). $\mathcal{P}$ is the smallest set such that whenever P, P_i are processes then $\mathcal{P}$ contains

(1) $\sum_{i\in I} S_i:\alpha_i.P_i$ (guarded summation: I finite, each S_i finite)
(2) $P_1 \mid P_2$ (parallel composition)
(3) $\mathsf{new}\ a\ P$ (restriction)
(4) $A\langle a_1, .., a_n\rangle$ (identifier)

$\mathcal{P}$ is ranged over by $P, Q, R, \ldots$ We let $M, N, \ldots$ range over (guarded) summations. We assume that each identifier A comes with a defining equation $A(a_1, \ldots, a_n) \stackrel{\text{def}}{=} P$, where $a_1, \ldots, a_n$ are all distinct and P is a process all of whose free names are drawn from $a_1, \ldots, a_n$. We will tend to abbreviate $a_1, \ldots, a_n$ by $\vec{a}$. We write the empty guarded summation as $\mathbf{0}$ and abbreviate $S:\alpha.\mathbf{0}$ by $S:\alpha$. It is assumed that the order in a summation is immaterial. We abbreviate $\emptyset:\alpha$ by α. The prefixed operations ($\mathsf{new}\ a$ and $S:\alpha.$) bind most tightly, followed by parallel composition and finally summation.

Definition 3.1 is much as in standard CCS, except for the priority guards S_i. The meaning of the priority guard $S:\alpha$ is that α can only be performed if the environment does not offer any action in $\overline{S} \cap \mathsf{Pri}$. Clearly, any actions in $S - \mathsf{Pri}$ have no effect as guards, and can be eliminated without changing the behaviour of a process. We allow them to occur in the syntax, since otherwise we could not freely instantiate the parameters in an identifier. We write $u:\alpha$ instead of $\{u\}:\alpha$. Restriction is a variable-binding operator, and we write $\mathsf{fn}(P)$ for the free names of P.

Definition 3.2 (*Free names*). We define $\mathsf{fn}(P) \subseteq \mathcal{N} \cup \mathcal{U}$ by induction on $P \in \mathcal{P}$:

$$\mathsf{fn}(\textstyle\sum_{i\in I} S_i:\alpha_i.P_i) = \{n \in \mathcal{N} \cup \mathcal{U} : \exists i \in I.n \in S_i \cup \{\alpha_i\} \vee \overline{n} \in S_i \cup \{\alpha_i\} \vee n \in \mathsf{fn}(P_i)\}$$
$$\mathsf{fn}(P_1 \mid P_2) = \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2)$$
$$\mathsf{fn}(\mathsf{new}\ a\ P) = \mathsf{fn}(P) - \{a\}$$
$$\mathsf{fn}(A\langle a_1, \ldots, a_n\rangle) = \{a_1, \ldots, a_n\}$$

Two sublanguages of CPG are of interest:

Definition 3.3. Let $\mathcal{P}_{\mathsf{Ug}}$ be the sublanguage of *unguarded* processes generated as in Definition 3.1 except that all priority guards are empty (i.e. $S_i = \emptyset$ in clause (1)).

Let $\mathcal{P}_{\mathsf{Std}}$ be the sublanguage of *standard* processes generated as in Definition 3.1 except that all priority guards are empty and all names (whether free or bound) are standard (i.e. belong to $\mathcal{N}$).

[1] We are following the formulation of CCS in [19] rather than that of [17]. Processes such as $P + (Q \mid R)$ are not allowed, only guarded choices $\sum_{i\in I} \alpha_i.P_i$.

Clearly $\mathcal{P}_{\mathsf{Std}} \subseteq \mathcal{P}_{\mathsf{Ug}} \subseteq \mathcal{P}$. Note that $\mathcal{P}_{\mathsf{Std}}$ is effectively standard CCS. The unguarded processes $\mathcal{P}_{\mathsf{Ug}}$ differ from $\mathcal{P}_{\mathsf{Std}}$ in that they may contain actions in Pri. Such processes cause no problems for strong equivalence (Proposition 6.5), but care is needed with weak equivalence (Section 7), since e.g. u and $\tau.u$ ($u \in \mathsf{Pri}$) are not weakly equivalent, as remarked above.

4. Offers and reaction

In this section we define the reaction relation. This will be parametrised on the set of priority actions offered by the environment.

Process contexts are defined much as in CCS:

Definition 4.1 (*cf.* [19, *Definition* 4.4]). A *process context* is given by

$$\mathcal{C} ::= [\cdot] \mid S{:}\alpha.\mathcal{C} + M \mid \mathsf{new}\, a\, \mathcal{C} \mid \mathcal{C} \mid P \mid P \mid \mathcal{C}$$

The *elementary contexts* are $S{:}\alpha.[\cdot] + M$, $\mathsf{new}\, a\, [\cdot]$, $[\cdot] \mid P$ and $P \mid [\cdot]$.

An equivalence relation is a *congruence* if it is preserved by all elementary contexts (which is equivalent to being preserved by all contexts).

Structural congruence is the most basic equivalence on processes, which facilitates reaction by bringing the subprocesses which are to react with each other into juxtaposition. It is defined as for CCS:

Definition 4.2 (*cf.* [19, *Definition* 4.7]). *Structural congruence*, written $\equiv$, is the congruence on $\mathcal{P}$ generated by the following equations:

(1) Change of bound names (alpha-conversion)
(2) Reordering of terms in a summation
(3) $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
(4) $\mathsf{new}\, a\, (P \mid Q) \equiv P \mid \mathsf{new}\, a\, Q$ if $a \notin \mathsf{fn}(P)$;
$\mathsf{new}\, a\, \mathbf{0} \equiv \mathbf{0}$, $\mathsf{new}\, a\, \mathsf{new}\, b\, P \equiv \mathsf{new}\, b\, \mathsf{new}\, a\, P$
(5) $A\langle \vec{b} \rangle \equiv \{\vec{b}/\vec{a}\}P$ if $A(\vec{a}) \stackrel{\text{def}}{=} P$

In item (1) we require that the distinction between standard names in $\mathcal{N}$ and priority names in $\mathcal{U}$ is observed: a bound name in $\mathcal{N}$ can only be replaced by another name in $\mathcal{N}$, and the same for $\mathcal{U}$.

To see why we forbid α-conversion between standard and priority names, consider the example $\mathsf{new}\, u\, (u{:}a \mid \overline{u})$. Here a cannot occur, since it is inhibited by its priority guard, as the environment is offering $\overline{u}$. But if we were permitted to α-convert u to a standard name b, we would get the process $\mathsf{new}\, b\, (b{:}a \mid \overline{b})$, which *can* perform a, since standard names have no effect in priority guards.

It is easy to see that $\mathcal{P}_{\mathsf{Ug}}$ is closed under $\equiv$; however $\mathcal{P}_{\mathsf{Std}}$ is not, because of the law $\mathsf{new}\, a\, \mathbf{0} \equiv \mathbf{0}$. If $a \in \mathcal{U}$ then we have $\mathbf{0} \in \mathcal{P}_{\mathsf{Std}}$, but $\mathsf{new}\, a\, \mathbf{0} \notin \mathcal{P}_{\mathsf{Std}}$. However it turns out that structural congruence when restricted to standard processes is just standard structural congruence. Let $\equiv_{\mathsf{Std}}$ be defined just as in Definition 4.2, except that we confine attention to processes and contexts in $\mathcal{P}_{\mathsf{Std}}$.

Proposition 4.3. *If* $P, Q \in \mathcal{P}_{\mathsf{Std}}$ *then* $P \equiv Q$ *iff* $P \equiv_{\mathsf{Std}} Q$.

Proof. ($\Rightarrow$) Suppose $P \equiv Q$. Then there is a chain $P = P_0 \equiv \cdots \equiv P_k = Q$, such that for each $i = 0, \ldots, k-1$ there are processes Q_i, R_i with $Q_i \equiv R_i$ or $R_i \equiv Q_i$ obtained by substitution from one of the generating equations of $\equiv$, and there is a context $\mathcal{C}_i$ with $P_i = \mathcal{C}_i[Q_i]$, $P_{i+1} = \mathcal{C}_i[R_i]$. Our task is to alter the chain in such a way that all the processes and contexts belong to $\mathcal{P}_{\mathsf{Std}}$. We work along the chain from P to Q. The only issue is the law $\mathsf{new}\, a\, \mathbf{0} \equiv \mathbf{0}$, when used from right to left. If for some i we have $Q_i = \mathbf{0}$ and $R_i = \mathsf{new}\, u\, \mathbf{0}$ (where $u \in \mathcal{U}$), then we alter R_i to $R_i' = \mathsf{new}\, a\, \mathbf{0}$, where $a \in \mathcal{N}$ is some fresh standard name. We furthermore change any subsequent related occurrence of $\mathsf{new}\, u$ to $\mathsf{new}\, a$. By this means we obtain a valid chain entirely within $\mathcal{P}_{\mathsf{Std}}$, showing that $P \equiv_{\mathsf{Std}} Q$.

($\Leftarrow$) Immediate from the definitions. □

Proposition 4.3 shows that structural congruence on $\mathcal{P}$ is conservative over the usual structural congruence on standard processes.

Recall that a guarded action $S:\alpha$ is conditional on other processes in the environment not offering actions in $\overline{S} \cap \mathsf{Pri}$. Before defining reaction we define for each process P the set $\mathsf{off}(P) \subseteq \mathsf{Pri}$ of "higher priority" actions "offered" by P.

Definition 4.4 (*Offers*). For $P \in \mathcal{P}$ and $u \in \mathsf{Pri}$ we define the relation $P \ \mathsf{off}\ u$ ("P offers u") by the following rules:

$$M + S:u.P + N \ \mathsf{off}\ u \text{ if } u \notin S$$

$$\frac{P \ \mathsf{off}\ u}{P \mid Q \ \mathsf{off}\ u} \qquad \frac{Q \ \mathsf{off}\ u}{P \mid Q \ \mathsf{off}\ u}$$

$$\frac{P \ \mathsf{off}\ u}{\mathsf{new}\ a\ P \ \mathsf{off}\ u} \text{ if } a \neq \mathsf{name}(u) \qquad \frac{\{\vec{b}/\vec{a}\}P \ \mathsf{off}\ u}{A\langle\vec{b}\rangle \ \mathsf{off}\ u} \text{ if } A(\vec{a}) \stackrel{\text{def}}{=} P$$

We furthermore define $\mathsf{off}(P) = \{u \in \mathsf{Pri} : P \ \mathsf{off}\ u\}$.

In the first rule, the reason that we insist $u \notin S$ is that we want to equate a process such as $u:u$ with $\mathbf{0}$, since $u:u$ can never engage in a reaction. Note that if $P \in \mathcal{P}_{\mathsf{Std}}$ then $\mathsf{off}(P) = \emptyset$.

A process can offer an action without being able to perform the corresponding transition. For instance, the process $u:v \mid \overline{u}$ offers v, but we shall see in Section 5 that it cannot perform v, since the environment $\overline{u}$ prevents this.

Lemma 4.5. *For any* $P \in \mathcal{P}$:
(1) $\mathsf{fn}(P)$ *is finite;*
(2) $\mathsf{off}(P) \subseteq (\mathsf{fn}(P) \cup \overline{\mathsf{fn}(P)}) \cap \mathsf{Pri}$;
(3) *if* $P \equiv Q$ *then* $\mathsf{fn}(P) = \mathsf{fn}(Q)$ *and* $\mathsf{off}(P) = \mathsf{off}(Q)$.

Proof. Straightforward and omitted. □

In CPG, a reaction can be conditional on offers from the environment. Consider $u:b \mid \overline{b}$. This can react by communication on b and $\overline{b}$. However b is guarded by u, and so the reaction is conditional on the environment not offering $\overline{u}$. We reflect this by letting reaction be parametrised on sets of actions $U \subseteq \mathsf{Pri}$. The intended meaning of $P \rightarrow_U P'$ is that P can react on its own, as long as the environment does not offer $\overline{u}$ for any $u \in U$ (in our parlance, "eschews" U).

Definition 4.6. Let $P \in \mathcal{P}$ and let $S \subseteq \mathsf{Vis}$ be finite. P *eschews* S (written $P \ \mathsf{eschews}\ S$) iff $\mathsf{off}(P) \cap \overline{S} = \emptyset$.

Proposition 4.7. *Let* $P \in \mathcal{P}$ *and let* $S \subseteq \mathsf{Vis}$ *be finite.*
(1) *If* $P \ \mathsf{eschews}\ S$ *and* $T \subseteq S$ *then* $P \ \mathsf{eschews}\ T$.
(2) *If* $P \ \mathsf{eschews}\ S$ *and* $Q \ \mathsf{eschews}\ S$ *then* $(P \mid Q) \ \mathsf{eschews}\ S$.

Definition 4.8 (*cf.* [19, *Definition* 4.13]). The *reaction relation* on $\mathcal{P}$ is the smallest relation $\rightarrow$ on $\mathcal{P} \times \wp(\mathsf{Pri}) \times \mathcal{P}$ generated by the following rules:

$$S:\tau.P + M \rightarrow_{S \cap \mathsf{Pri}} P$$

$$\frac{S:a.P + M \ \mathsf{eschews}\ T \quad T:\overline{a}.Q + N \ \mathsf{eschews}\ S}{(S:a.P + M) \mid (T:\overline{a}.Q + N) \rightarrow_{(S \cup T) \cap \mathsf{Pri}} P \mid Q} \qquad \frac{P \rightarrow_U P' \quad Q \ \mathsf{eschews}\ U}{P \mid Q \rightarrow_U P' \mid Q}$$

$$\frac{P \rightarrow_U P'}{\mathsf{new}\ a\ P \rightarrow_{U - \{a,\overline{a}\}} \mathsf{new}\ a\ P'} \qquad \frac{Q \equiv P \quad P \rightarrow_U P' \quad P' \equiv Q'}{Q \rightarrow_U Q'}$$

We abbreviate $P \rightarrow_\emptyset P'$ by $P \rightarrow P'$. A reaction $P \rightarrow P'$ is said to be *unconditional*.

The second clause of Definition 4.8 is the most important. In order for an action a to react with a complementary $\overline{a}$, the two sides must not preempt each other (i.e. they must eschew each other's guards). Furthermore the reaction remains conditional on the environment eschewing the union of their guards. The restriction rule shows how this conditionality can then be removed by scoping. Notice that if we restrict attention to the unguarded processes $\mathcal{P}_{\mathsf{Ug}}$ then whenever $P \rightarrow_U P'$ we have $U = \emptyset$, and we recover the usual CCS reaction relation. So the new reaction relation is conservative over the old.

Reactions cannot introduce new free names:

Lemma 4.9. *If* $P \rightarrow_U P'$ *then* $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$.

Proof. Straightforward and omitted. □

Lemma 4.9 will be useful when we consider networks and electoral systems in Section 9.

5. Labelled transitions

As in standard CCS, we wish to define a transition relation on processes $P \xrightarrow{\alpha} P'$ meaning that P can perform action α and become P'. As we did with reaction, we refine the transition relation so that it is parametrised on sets of actions $U \subseteq \mathsf{Pri}$. The intended meaning of $P \xrightarrow{\alpha}_U P'$ is that P can perform α as long as the environment eschews U. Our definition is inspired by the transition relation in [5], which is parametrised on what set of output actions the environment is ready to perform.

Definition 5.1 (*cf.* [19, *Definition* 5.1]). The *transition relation* on $\mathcal{P}$ is the smallest relation $\rightarrow$ on $\mathcal{P} \times \mathsf{Act} \times \wp(\mathsf{Pri}) \times \mathcal{P}$ generated by the following rules:

$$\text{(sum)} \qquad M + S{:}\alpha.P + N \xrightarrow{\alpha}_{S \cap \mathsf{Pri}} P \quad \text{if } \alpha \notin S \cap \mathsf{Pri}$$

$$\text{(react)} \qquad \frac{P_1 \xrightarrow{\lambda}_{U_1} P_1' \quad P_2 \xrightarrow{\overline{\lambda}}_{U_2} P_2' \quad P_1 \text{ eschews } U_2 \quad P_2 \text{ eschews } U_1}{P_1 \mid P_2 \xrightarrow{\tau}_{U_1 \cup U_2} P_1' \mid P_2'}$$

$$\text{(par)} \qquad \frac{P_1 \xrightarrow{\alpha}_U P_1' \quad P_2 \text{ eschews } U}{P_1 \mid P_2 \xrightarrow{\alpha}_U P_1' \mid P_2} \qquad \frac{P_2 \xrightarrow{\alpha}_U P_2' \quad P_1 \text{ eschews } U}{P_1 \mid P_2 \xrightarrow{\alpha}_U P_1 \mid P_2'}$$

$$\text{(res)} \qquad \frac{P \xrightarrow{\alpha}_U P'}{\mathsf{new}\, a\ P \xrightarrow{\alpha}_{U - \{a, \overline{a}\}} \mathsf{new}\, a\ P'} \quad \text{if } \alpha \notin \{a, \overline{a}\}$$

$$\text{(ident)} \qquad \frac{\{\vec{b}/\vec{a}\}P \xrightarrow{\alpha}_U P'}{A\langle \vec{b} \rangle \xrightarrow{\alpha}_U P'} \quad \text{if } A(\vec{a}) \stackrel{\text{def}}{=} P$$

We abbreviate $P \xrightarrow{\alpha}_{\emptyset} P'$ by $P \xrightarrow{\alpha} P'$ and $\exists P'.P \xrightarrow{\alpha}_U P'$ by $P \xrightarrow{\alpha}_U$.

Proposition 5.2. *Let* $P, P' \in \mathcal{P}$, $\alpha \in \mathsf{Act}$ *and* $U \subseteq \mathsf{Pri}$.

(1) *If* $P \xrightarrow{\alpha}_U P'$ *then*
- $\alpha \in \mathsf{fn}(P)$;
- $\alpha \notin U$ *and* U *is finite; and*
- *if* $u \in U$ *then* $\mathsf{name}(u) \in \mathsf{fn}(P)$ *and* u *belongs to some priority guard occurring in* P.

(2) *If* $P \xrightarrow{u}_U$ *then* P off u.

Proof. Straightforward and omitted. □

To see that the converse to Proposition 5.2(2) does not hold in general, consider $u:v \mid \overline{u}$. As we noted in Section 4, $u:v \mid \overline{u}$ off v but $u:v \mid \overline{u}$ cannot perform v.

As with reaction, note that if we restrict attention to the unguarded processes $\mathcal{P}_{\mathsf{Ug}}$ then whenever $P \xrightarrow{\alpha}_U P'$ we have $U = \emptyset$, so that we recover the usual CCS transition relation. Thus the new transition relation is conservative over the old. In applications we envisage that the standard CCS transition relation can be used most of the time. The CPG transition relation will only be needed in those subsystems which use priority.

As an illustration of the design choices embodied in our definitions, consider the circular example of Camilleri and Winskel (Subsection 2.1):

$$P \stackrel{\text{def}}{=} u.a + u:\overline{v} \quad Q \stackrel{\text{def}}{=} v.b + v:\overline{u} \quad R \stackrel{\text{def}}{=} \mathsf{new}\, u, v\, (P \mid Q)$$

In P action u has priority over $\overline{v}$, while in Q action v has priority over $\overline{u}$. We have $P \xrightarrow{u} a$, $Q \xrightarrow{\overline{u}}_v \mathbf{0}$. For a u communication to happen, by rule (react) we need $\overline{v} \notin \mathsf{off}(P)$, but $\mathsf{off}(P) = \{u, \overline{v}\}$, so that the u communication cannot happen. Similarly the v communication cannot happen, and so R is strongly equivalent to $\mathbf{0}$ (strong offer equivalence is defined in the next section).

Structural congruence respects transition:

Lemma 5.3 (cf. [19, Lemma 5.2]). *Let $P, P' \in \mathcal{P}$, $\alpha \in \mathsf{Act}$, $U \subseteq \mathsf{Pri}$. If $P \equiv \xrightarrow{\alpha}_U P'$ then $P \xrightarrow{\alpha}_U \equiv P'$.*

Proof. Let $\mathcal{C}[\cdot]$ be a process context. We must show that if $P \equiv Q$ is a generating case of Definition 4.2, and $\mathcal{C}[Q] \xrightarrow{\alpha}_U P'$ and then $\mathcal{C}[P] \xrightarrow{\alpha}_U Q' \equiv P'$, for some Q'. The proof is by cases on the rules for $\rightarrow$ and the generating equations of $\equiv$. The interesting case is the law $(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$. We consider two example transitions of $(P_1 \mid P_2) \mid P_3$, and omit the many other similar cases.

Suppose first that P_1 moves on its own in $(P_1 \mid P_2) \mid P_3$, so that

$$(P_1 \mid P_2) \mid P_3 \xrightarrow{\alpha}_U (P_1' \mid P_2) \mid P_3$$

We have $P_1 \xrightarrow{\alpha}_U P_1'$ and so P_2 $\mathsf{eschews}$ U and P_3 $\mathsf{eschews}$ U. By Proposition 4.7 we deduce that $(P_2 \mid P_3)$ $\mathsf{eschews}$ U. Hence

$$P_1 \mid (P_2 \mid P_3) \xrightarrow{\alpha}_U P_1' \mid (P_2 \mid P_3)$$

Now suppose that $P_1 \xrightarrow{a}_U P_1'$ and $P_2 \xrightarrow{\overline{a}}_V P_2'$, and that

$$(P_1 \mid P_2) \mid P_3 \xrightarrow{\tau}_{U \cup V} (P_1' \mid P_2') \mid P_3$$

We have P_1 $\mathsf{eschews}$ V, P_2 $\mathsf{eschews}$ U and P_3 $\mathsf{eschews}$ $U \cup V$. By Proposition 4.7 we deduce that P_3 $\mathsf{eschews}$ V and $(P_2 \mid P_3)$ $\mathsf{eschews}$ U. Therefore $P_2 \mid P_3 \xrightarrow{\overline{a}}_V P_2' \mid P_3$, and

$$P_1 \mid (P_2 \mid P_3) \xrightarrow{\tau}_{U \cup V} P_1' \mid (P_2' \mid P_3) \qquad \square$$

Lemma 5.4 (cf. [19, Lemma 5.5]). *Let $P, P' \in \mathcal{P}$, $\lambda \in \mathsf{Vis}$, $U \subseteq \mathsf{Pri}$. If $P \xrightarrow{\lambda}_U P'$ then P and P' can be expressed, up to $\equiv$, as*

$$P \equiv \mathsf{new}\, \vec{a}\, ((S:\lambda.Q + M) \mid R) \qquad P' \equiv \mathsf{new}\, \vec{a}\, (Q \mid R)$$

(some $\vec{a}$, S, Q, M, R) with $\lambda, \overline{\lambda} \notin \vec{a}$ and $U = S \cap \mathsf{Pri}$.

Proof. By induction on the derivation of $P \xrightarrow{\lambda}_U P'$. We omit the details. $\square$

Silent transitions agree with reaction:

Proposition 5.5 (cf. [19, Lemma 5.6]). *Let $P, P' \in \mathcal{P}$, $U \subseteq \mathsf{Pri}$. Then $P \xrightarrow{\tau}_U \equiv P'$ iff $P \rightarrow_U P'$.*

Proof. ($\Rightarrow$) It is enough to show that $P \xrightarrow{\tau}_U P'$ implies $P \rightarrow_U \equiv P'$. By induction on the proof of $P \xrightarrow{\tau}_U P'$. For case (react) we employ Lemma 5.4. We omit the details.

($\Leftarrow$) By induction on the proof of $P \rightarrow_U P'$. For the rule for structural congruence we use Lemma 5.3. We omit the details. □

Labelled transitions cannot introduce new free names (cf. Lemma 4.9):

Lemma 5.6. *If* $P \xrightarrow{\alpha}_U P'$ *then* $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$.

Proof. Straightforward and omitted. □

Lemma 5.6 will be used in Sections 6 and 9.

6. Strong offer bisimulation

Similarly to standard CCS, we shall define process equivalences based on strong and weak bisimulation. We consider strong bisimulation in this section and weak bisimulation (i.e. with hiding of silent actions) in the next. We shall give a set of axioms which is complete for strong bisimulation on finite processes (Theorem 6.14). We shall also show that in certain circumstances priority guards can be used together with parallel composition to encode summation (Proposition 6.15).

As we said in Section 1, for processes to be equivalent they must make the same offers, and for a process Q to simulate a process P, Q must be able to do whatever P can, though possibly constrained by fewer or smaller priority guards. Thus $a + u:a$ and a will be equivalent, since $u:a$ is simulated by a.

Definition 6.1 (*cf.* [19]). A symmetric relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a *strong offer bisimulation* if $\mathcal{S}(P, Q)$ implies both that $\mathsf{off}(P) = \mathsf{off}(Q)$ and that for all $\alpha \in \mathsf{Act}$, if $P \xrightarrow{\alpha}_U P'$ then for some Q' and $V \subseteq U$, we have $Q \xrightarrow{\alpha}_V Q'$ and $\mathcal{S}(P', Q')$. Processes P and Q are *strongly offer equivalent*, written $P \overset{\mathsf{off}}{\sim} Q$, iff there is some strong offer bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.

In view of the general theory of bisimulation [19, Section 3.3], $\overset{\mathsf{off}}{\sim}$ is an equivalence relation, and is itself a strong offer bisimulation.

Proposition 6.2 (cf. [19, Theorem 5.13]). $\equiv$ *is a strong offer bisimulation. Hence* $\equiv$ *implies* $\overset{\mathsf{off}}{\sim}$.

Proof. By Lemma 5.3. □

Theorem 6.3 (cf. [19, Proposition 5.29]). *Strong offer equivalence is a congruence.*

Proof. The case of parallel composition is the most interesting. Suppose $P \overset{\mathsf{off}}{\sim} Q$. We must show $P \mid R \overset{\mathsf{off}}{\sim} Q \mid R$.

Clearly if $\mathsf{off}(Q) = \mathsf{off}(P)$ then $\mathsf{off}(Q \mid R) = \mathsf{off}(P \mid R)$.

Suppose $P \mid R \xrightarrow{\alpha}_U P' \mid R$ is derived from $P \xrightarrow{\alpha}_U P'$. Then R $\mathsf{eschews}$ U and $Q \xrightarrow{\alpha}_{U'} Q'$ with $U' \subseteq U$ and $P' \overset{\mathsf{off}}{\sim} Q'$. So clearly R $\mathsf{eschews}$ U' and $Q \mid R \xrightarrow{\alpha}_{U'} Q' \mid R$.

Now suppose $P \mid R \xrightarrow{\alpha}_U P \mid R'$ is derived from $R \xrightarrow{\alpha}_U R'$. Then P $\mathsf{eschews}$ U. So Q $\mathsf{eschews}$ U and we have $Q \mid R \xrightarrow{\alpha}_U Q \mid R'$.

Now suppose $P \mid R \xrightarrow{\tau}_{U \cup V} P' \mid R'$ is derived from $P \xrightarrow{\lambda}_U P'$ and $R \xrightarrow{\overline{\lambda}}_V R'$. So P $\mathsf{eschews}$ V and R $\mathsf{eschews}$ U. Then $Q \xrightarrow{\lambda}_{U'} Q'$ with $U' \subseteq U$ and $P' \overset{\mathsf{off}}{\sim} Q'$. Since $\mathsf{off}(Q) = \mathsf{off}(P)$, Q $\mathsf{eschews}$ V. Also R $\mathsf{eschews}$ U'. Hence $Q \mid R \xrightarrow{\tau}_{U' \cup V} Q' \mid R'$.

The cases for summation, restriction, and identifier are straightforward and omitted. □

For unguarded processes, strong offer equivalence coincides with strong equivalence in the usual sense of [19] (using the full set of actions Act, including priority actions).

Definition 6.4 (*cf.* [19]). A symmetric relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a *strong bisimulation* if $\mathcal{S}(P, Q)$ implies that for all $\alpha \in \mathsf{Act}$, if $P \xrightarrow{\alpha} P'$ then for some Q', we have $Q \xrightarrow{\alpha} Q'$ and $\mathcal{S}(P', Q')$. Processes P and Q are *strongly equivalent*, written $P \sim Q$, iff there is some strong bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.

Proposition 6.5. *Let* $P, Q \in \mathcal{P}_{\mathsf{Ug}}$. *Then* $P \overset{\mathsf{off}}{\sim} Q$ *iff* $P \sim Q$.

Proof. ($\Rightarrow$) Trivial.

($\Leftarrow$) We show that $\sim$ is a strong offer bisimulation on $\mathcal{P}_{\mathsf{Ug}}$. It is enough to show that for $P, Q \in \mathcal{P}_{\mathsf{Ug}}$, if $P \sim Q$ then $\mathsf{off}(P) = \mathsf{off}(Q)$. For any $R \in \mathcal{P}_{\mathsf{Ug}}$ recall that if $R \xrightarrow{\alpha}_U$ then $U = \emptyset$. It is straightforward to see that $\mathsf{off}(R) = \{u \in \mathsf{Pri} : R \xrightarrow{u}\}$. Take $P, Q \in \mathcal{P}_{\mathsf{Ug}}$. If $P \sim Q$ then $\{u \in \mathsf{Pri} : P \xrightarrow{u}\} = \{u \in \mathsf{Pri} : Q \xrightarrow{u}\}$. Hence $\mathsf{off}(P) = \mathsf{off}(Q)$. □

Plainly $\sim$ is exactly standard strong equivalence when restricted to standard processes. So $\overset{\mathsf{off}}{\sim}$ is conservative over $\sim$ and we can reuse all the known equivalences between CCS processes when working with CPG processes.

Much as in CCS, every process is equivalent to a summation:

Proposition 6.6 (cf. [19, Proposition 5.21]). *For all* $P \in \mathcal{P}$,

$$P \overset{\mathsf{off}}{\sim} \sum\{U : \alpha.Q : P \xrightarrow{\alpha}_U Q\}$$

Proof. Similar to the proof of [19, Proposition 5.21]. □

A key design feature of CPG is that priority can be localised with the use of restriction. This is demonstrated by the next result, which states that a process where all priority names are restricted is equivalent to a standard CCS process.

Proposition 6.7. *Let* P *be a CPG process such that* $\mathsf{fn}(P) \subseteq \mathcal{N}$. *Then there is* $Q \in \mathcal{P}_{\mathsf{Std}}$ *such that* $P \overset{\mathsf{off}}{\sim} Q$.

Proof. Suppose $\mathsf{fn}(P) \subseteq \mathcal{N}$. If $\mathsf{fn}(Q) \subseteq \mathcal{N}$ and $Q \xrightarrow{\alpha}_U Q'$ then $\mathsf{fn}(Q') \subseteq \mathcal{N}$ by Lemma 5.6, and $\alpha \in \mathsf{Std}$ and $U = \emptyset$ by Proposition 5.2(1). For every Q such that $\mathsf{fn}(Q) \subseteq \mathcal{N}$, let A_Q be a new constant with defining equation

$$A_Q \overset{\mathrm{def}}{=} \sum\{\alpha.A_{Q'} : Q \xrightarrow{\alpha} Q'\}$$

By Proposition 6.6 we have $Q \overset{\mathsf{off}}{\sim} A_Q$. Also $A_Q \in \mathcal{P}_{\mathsf{Std}}$. In particular, $P \overset{\mathsf{off}}{\sim} A_P$ and $A_P \in \mathcal{P}_{\mathsf{Std}}$, as required. □

In similar vein, a further consequence of Propositions 6.6 and 5.2(1) is that if all priority names belonging to priority guards of a finite process P are bound, then $P \overset{\mathsf{off}}{\sim} Q$ for some finite $Q \in \mathcal{P}_{\mathsf{Ug}}$. Thus a finite process where all priority guards are bound is equivalent to a process with no priority guards.

Proposition 6.8. *The following laws hold:*

$$M + S:\alpha.P \overset{\mathsf{off}}{\sim} M + (S \cap \mathsf{Pri}):\alpha.P \tag{1}$$

$$M + U:\alpha.P \overset{\mathsf{off}}{\sim} M \quad \text{if } \alpha \in U \subseteq \mathsf{Pri} \tag{2}$$

$$M + U:\alpha.P + (U \cup V):\alpha.P \overset{\mathsf{off}}{\sim} M + U:\alpha.P \tag{3}$$

$$\left(\sum_{i\in I} U_i:\alpha_i.P_i\right) \Big| \left(\sum_{j\in J} V_j:\beta_j.Q_j\right)$$

$$\stackrel{\text{off}}{\sim} \sum_{i\in I}\left\{U_i:\alpha_i.\left(P_i \Big| \left(\sum_{j\in J} V_j:\beta_j.Q_j\right)\right) : \forall j\in J.\beta_j \notin \overline{U}_i\right\} \tag{4}$$

$$+\sum_{j\in J}\left\{V_j:\beta_j.\left(\left(\sum_{i\in I} U_i:\alpha_i.P_i\right)\Big| Q_j\right) : \forall i\in I.\alpha_i \notin \overline{V}_j\right\}$$

$$+\sum_{i\in I, j\in J}\{(U_i\cup V_j):\tau.(P_i \mid Q_j) : \alpha_i = \overline{\beta}_j \in \mathsf{Vis}$$

$$\wedge\, \forall i'\in I\, \forall j'\in J(\alpha_{i'}\notin \overline{V}_j \wedge \beta_{j'}\notin \overline{U}_i)\}$$

$$\mathsf{new}\ a\left(\sum_{i\in I} U_i:\alpha_i.P_i\right) \stackrel{\text{off}}{\sim} \sum_{i\in I}(U_i-\{a,\overline{a}\}):\alpha_i.\mathsf{new}\ a\ P_i : \alpha_i\neq\{a,\overline{a}\} \tag{5}$$

Proof. Straightforward and omitted. □

Laws (1) and (2) are trivial, and correspond to basic features of the transition system. Law (3) is the most interesting, and expresses the essential nature of offer bisimulation: the extra behaviour $(U\cup V):\alpha.P$ of the left-hand side can be simulated by the stronger behaviour $U:\alpha.P$ of the right-hand side. Laws (4) and (5) are the CPG version of the Expansion Law of CCS [19, Proposition 5.23].

Definition 6.9. Let $\mathcal{A}_S$ be the following set of axioms: the axioms of structural congruence $\equiv$ (Definition 4.2) together with the five laws of Proposition 6.8.

Clearly $\mathcal{A}_S$ is sound for strong offer equivalence by Propositions 6.2 and 6.8. We shall show that it is complete for finite processes (a CPG process is *finite* if it contains no identifiers). First we define a notion of depth which will be useful in the proof.

Definition 6.10. The *depth* of a finite process $P\in\mathcal{P}$ is defined by induction as follows:

$$\mathsf{depth}(\mathbf{0}) \stackrel{\text{def}}{=} 0$$
$$\mathsf{depth}(\textstyle\sum_{i\in I} S_i:\alpha_i.P_i) \stackrel{\text{def}}{=} 1+\max\{\mathsf{depth}(P_i) : i\in I\} \quad (I\neq\emptyset)$$
$$\mathsf{depth}(P\mid Q) \stackrel{\text{def}}{=} \mathsf{depth}(P)+\mathsf{depth}(Q)$$
$$\mathsf{depth}(\mathsf{new}\ a\ P) \stackrel{\text{def}}{=} \mathsf{depth}(P)$$

Definition 6.11. P is in *standard form* if $P\stackrel{\text{id}}{=}\sum_{i\in I} U_i:\alpha_i.P_i$ where for each $i\in I$ we have $\alpha_i\notin U_i$, $U_i\subseteq\mathsf{Pri}$ and P_i is in standard form. Here $\stackrel{\text{id}}{=}$ means "identically equal".

Lemma 6.12. *For any finite process P there is P' in standard form such that* $\mathsf{depth}(P')\leq\mathsf{depth}(P)$ *and* $\mathcal{A}_S\vdash P=P'$.

Proof. First of all, it is straightforward to check that for finite processes P, Q,

- if $P\equiv Q$ then $\mathsf{depth}(P)=\mathsf{depth}(Q)$; and
- if $P=Q$ is deduced by applying one of the five laws of Proposition 6.8 from left to right, then $\mathsf{depth}(P)\geq\mathsf{depth}(Q)$.

In fact, laws (1), (3) and (5) preserve depth, while laws (2) and (4) may preserve or reduce depth when applied from left to right. In this proof we shall always apply the five laws from left to right.

Suppose for a contradiction that there is some finite P which has no equivalent standard form. We can take P to be of minimal depth. Also, we can assume that any subterm of P (even if of the same depth as P) can be put in standard form without increasing depth. We show that P can in fact be put in standard form, which will be a contradiction. There are three cases:

(1) $P \stackrel{\text{id}}{=} \sum_{i\in I} S_i:\alpha_i.P_i$. Since the P_i are subterms of P, we can convert the P_i to standard forms P_i' using $\mathcal{A}_S$, and without increasing depth. We now convert $\sum_{i\in I} S_i:\alpha_i.P_i'$ to standard form using laws (1), (2) and (3) and reordering of summation.

(2) $P \stackrel{\text{id}}{=} P_1 \mid P_2$. Since P_1 and P_2 are subterms of P, we can convert them to standard forms P_1' and P_2' without increasing depth. We now use law (4) to convert $P_1' \mid P_2'$ to the form $\sum_{i\in I} .U_i:\alpha_i.Q_i$. Here for each $i \in I$, $\mathsf{depth}(Q_i) < \mathsf{depth}(P_1' \mid P_2') \leq \mathsf{depth}(P)$, and so Q_i can be converted to standard form without increasing depth. We complete the conversion of P to standard form as in case (1).

(3) $P \stackrel{\text{id}}{=} \mathsf{new}\, a\ Q$. Since Q is a subterm of P we can convert it to standard form Q' without increasing depth. We now use law (5) to convert $\mathsf{new}\, a\ Q'$ to the form $\sum_{i\in I} .U_i:\alpha_i.Q_i$. Again the Q_i are all of lower depth than P. We now proceed as in case (2). □

Note that in the proof of Lemma 6.12 we only employed structural congruence in order to reorder summations.

Lemma 6.13. *If P is in standard form, $P \stackrel{\alpha}{\to}_U P'$ and $U \subseteq V$ then*

$$\mathcal{A}_S \vdash P = P + V:\alpha.P'$$

Proof. Use law (3) and reordering of summation. □

Theorem 6.14. *The set of axioms $\mathcal{A}_S$ is complete for $\stackrel{\text{off}}{\sim}$ on finite CPG processes.*

Proof. Given two processes $P \stackrel{\text{off}}{\sim} Q$ (which can both be taken to be in standard form by Lemma 6.12), we prove them equal by adding to Q a new summand for each summand of P (obtaining $Q + N$), and adding to P a new summand for each summand of Q (obtaining $P + M$).

Suppose $P \stackrel{\text{id}}{=} \sum_{i\in I} U_i:\alpha_i.P_i$. For each $i \in I$, we have $P \stackrel{\alpha_i}{\to}_{U_i} P_i$. So $Q \stackrel{\alpha_i}{\to}_{V_i} Q_i'$, for some $V_i \subseteq U_i$, and some Q_i' such that $P_i \stackrel{\text{off}}{\sim} Q_i'$. By Lemma 6.13, $\mathcal{A}_S \vdash Q = Q + U_i:\alpha_i.Q_i'$. By induction on the total depth of P and Q, we have $\mathcal{A}_S \vdash P_i = Q_i'$. Let $N = \sum_{i\in I} U_i:\alpha_i.Q_i'$. Then $\mathcal{A}_S \vdash Q = Q + N$. We get M symmetrically, with $\mathcal{A}_S \vdash P = P + M$. Now $\mathcal{A}_S \vdash P + M = Q + N$ and hence $\mathcal{A}_S \vdash P = Q$ as required. □

As mentioned in Section 1, in some circumstances we can encode mixed input and output guarded summation using priority guards and restricted parallel composition. The intuition is that committing to a particular choice entails preempting other choices, with this preemption being achieved through the use of priority.

Proposition 6.15. *Let $P \stackrel{\text{def}}{=} \sum_{i\in I} S_i:\alpha_i.P_i$ be a "non-self-dual" summation, i.e. there do not exist i, $j \in I$ and $\lambda \in \mathsf{Vis}$ such that $\alpha_i = \lambda$ and $\alpha_j = \overline{\lambda}$. Suppose also that $\mathsf{off}(P) = \emptyset$. Then*

$$P \stackrel{\text{off}}{\sim} \mathsf{new}\, u\ \left(\prod_{i\in I}(S_i \cup \{u\}:\alpha_i.(P_i \mid \overline{u}))\right)$$

where $u \in \mathcal{U}$ is some fresh name not occurring in P and $\prod$ denotes parallel composition.

Proof. Let $Q \stackrel{\text{def}}{=} \mathsf{new}\, u\ (\prod_{i\in I}(S_i \cup \{u\}:\alpha_i.(P_i \mid \overline{u})))$. We must show that $P \stackrel{\text{off}}{\sim} Q$. First of all we observe that $\mathsf{off}(P) = \mathsf{off}(Q) = \emptyset$. For each $i \in I$ let $Q_i \stackrel{\text{def}}{=} \prod_{j\in I, j\neq i}(S_j \cup \{u\}:\alpha_j.(P_j \mid \overline{u}))$. The derivatives of P are $P \stackrel{\alpha_i}{\to}_{S_i\cap \mathsf{Pri}} P_i$ for each $i \in I$. Clearly the derivatives of Q are $Q \stackrel{\alpha_i}{\to}_{S_i\cap \mathsf{Pri}} \equiv \mathsf{new}\, u\ (Q_i \mid P_i \mid \overline{u})$ for each $i \in I$ (here we use the non-self-dual property, and we use $\equiv$ to reorder the parallel composition for notational convenience). It is therefore enough

(with the use of Proposition 6.2) to show that $P_i \overset{\mathsf{off}}{\sim} \mathsf{new}\, u\, (Q_i \mid P_i \mid \overline{u})$ for each $i \in I$. This is done by showing that $\mathcal{S}_i = \{(P, \mathsf{new}\, u\, (Q_i \mid P \mid \overline{u})) : P \in \mathcal{P}, u \notin \mathsf{fn}(P)\}$ is a strong offer bisimulation for each $i \in I$ (here we need $\mathsf{off}(Q_i) = \emptyset$, which comes from $\mathsf{off}(P) = \emptyset$). We omit the checks that $\mathcal{S}_i$ is as stated. □

The non-self-dual condition in Proposition 6.15 is needed, since otherwise the right-hand side would have extra unwanted reactions. The condition is not unduly restrictive, since if we have a system where the same channel a is used to pass messages both to and from a process, we can simply separate a out into two separate channels, one for each direction. As an application of Proposition 6.15, notice that if $P \in \mathcal{P}_{\mathsf{Std}}$ is a non-self-dual summation then we can use the result to eliminate the (outermost) summation in favour of a parallel composition.

Remark 6.16. Milner [16] provides a complete axiomatisation for finite-state processes (with recursion) with respect to strong equivalence, by adding axioms to handle recursion. We conjecture that finite-state CPG processes can be axiomatised in exactly the same way.

7. Weak offer bisimulation

We now investigate weak bisimulation, where reactions are hidden. We shall give a set of axioms which are complete for weak bisimulation on finite processes (Theorem 7.10).

We start by defining weak transitions:

Definition 7.1. $P \Rightarrow_U P'$ iff $P \overset{\mathrm{id}}{=} P'$ or $\exists U_1, \ldots, U_n . P \overset{\tau}{\to}_{U_1} \cdots \overset{\tau}{\to}_{U_n} P'$ with $U = U_1 \cup \cdots \cup U_n$ $(n \geq 1)$. We abbreviate $P \Rightarrow_\emptyset P'$ by $P \Rightarrow P'$.

For $\lambda \in \mathsf{Vis}$, $P \overset{\lambda}{\Rightarrow}_U P'$ iff $\exists P'', P''', U', U'' . P \Rightarrow_{U'} P'' \overset{\lambda}{\to}_{U''} P''' \Rightarrow P'$ with $U = U' \cup U''$ and $\mathsf{off}(P'') \subseteq \mathsf{off}(P)$.

Recall that $P \overset{\mathrm{id}}{=} P'$ means that P and P' are identically equal. So $P \Rightarrow_U P'$ allows zero or more internal transitions with guards included in U. The condition $\mathsf{off}(P'') \subseteq \mathsf{off}(P)$ is needed to obtain a weak equivalence which is a congruence—we discuss this further below (after Theorem 7.4). The reason why we allow priority guards before performing a visible action, but not after, is as follows: for Q to simulate $P \overset{a}{\to}_U P'$, Q must expect an environment offering $\overline{U}$ up to and including performing a. After this, the environment has changed, and might be offering anything. So Q can perform further reactions to reach Q' simulating P', but these reactions must not be subject to any priority guards.

Definition 7.2. A symmetric relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a *weak offer bisimulation* if whenever $\mathcal{S}(P, Q)$ we have:

- $\mathsf{off}(P) = \mathsf{off}(Q)$; and
- if $P \overset{\tau}{\to}_U P'$ then for some Q' and $U' \subseteq U$, we have $Q \Rightarrow_{U'} Q'$ and $\mathcal{S}(P', Q')$; and
- for all $\lambda \in \mathsf{Vis}$, if $P \overset{\lambda}{\to}_U P'$ then for some Q' and $U' \subseteq U$, we have $Q \overset{\lambda}{\Rightarrow}_{U'} Q'$ and $\mathcal{S}(P', Q')$.

Processes P and Q are *weakly offer equivalent*, written $P \overset{\mathsf{off}}{\approx} Q$, iff there is some weak offer bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.

On the sublanguage $\mathcal{P}_{\mathsf{Std}}$ of standard processes (which corresponds to CCS), weak offer equivalence is the same as CCS weak equivalence (from [19, Proposition 6.3]).

As with strong offer equivalence, $\overset{\mathsf{off}}{\approx}$ is an equivalence relation, and is itself a weak offer bisimulation. Moreover, weak offer equivalence is entailed by strong offer equivalence:

Proposition 7.3. *For any P, Q, if $P \overset{\mathsf{off}}{\sim} Q$ then $P \overset{\mathsf{off}}{\approx} Q$.*

Proof. It is easy to check that any strong offer bisimulation is a weak offer bisimulation. □

Theorem 7.4 (cf. [19, Proposition 6.17]). $\overset{\mathsf{off}}{\approx}$ *is a congruence.*

Proof. Parallel composition is the most interesting. Suppose $P \overset{\mathsf{off}}{\approx} Q$. We must show $P \mid R \overset{\mathsf{off}}{\approx} Q \mid R$.

Clearly if $\mathsf{off}(P) = \mathsf{off}(Q)$ then $\mathsf{off}(P \mid R) = \mathsf{off}(Q \mid R)$.

Suppose $P \mid R \xrightarrow{\lambda}_U P' \mid R$ is derived from $P \xrightarrow{\lambda}_U P'$. Then R eschews U and $Q \Rightarrow_{U'} Q'' \xrightarrow{\lambda}_{U''} Q''' \Rightarrow Q'$, with $U', U'' \subseteq U$, $\mathsf{off}(Q'') \subseteq \mathsf{off}(Q)$ and $P' \overset{\mathsf{off}}{\approx} Q'$. So clearly R eschews U', R eschews U'' and $Q \mid R \Rightarrow_{U'} Q'' \mid R \xrightarrow{\lambda}_{U''} Q''' \mid R \Rightarrow Q' \mid R$ with $\mathsf{off}(Q'' \mid R) \subseteq \mathsf{off}(P \mid R)$. Thus $Q \mid R \overset{\lambda}{\Rightarrow}_{U' \cup U''} Q' \mid R$.

Suppose $P \mid R \xrightarrow{\tau}_U P' \mid R$ is derived from $P \xrightarrow{\tau}_U P'$. Then R eschews U and $Q \Rightarrow_{U'} Q'$ with $U' \subseteq U$. So clearly $Q \mid R \Rightarrow_{U'} Q' \mid R$.

Now suppose $P \mid R \xrightarrow{\alpha}_U P \mid R'$ is derived from $R \xrightarrow{\alpha}_U R'$. Then P eschews U. So since $\mathsf{off}(Q) = \mathsf{off}(P)$ we have Q eschews U and $Q \mid R \xrightarrow{\alpha}_U Q \mid R'$.

Finally suppose $P \mid R \xrightarrow{\tau}_{U \cup V} P' \mid R'$ is derived from $P \xrightarrow{\lambda}_U P'$ and $R \xrightarrow{\overline{\lambda}}_V R'$. So P eschews V and R eschews U. Then $Q \Rightarrow_{U'} Q'' \xrightarrow{\lambda}_{U''} Q''' \Rightarrow Q'$, with $U', U'' \subseteq U$, $\mathsf{off}(Q'') \subseteq \mathsf{off}(Q)$ and $P' \overset{\mathsf{off}}{\approx} Q'$. Then R eschews U' and so $Q \mid R \Rightarrow_{U'} Q'' \mid R$. Since $\mathsf{off}(Q'') \subseteq \mathsf{off}(Q) = \mathsf{off}(P)$ we see that Q'' eschews V. Also R eschews U''. So $Q'' \mid R \xrightarrow{\tau}_{U'' \cup V} Q''' \mid R' \Rightarrow Q' \mid R'$. Hence $Q \mid R \Rightarrow_{U' \cup U'' \cup V} Q' \mid R$.

Summation, restriction, identifier are straightforward and omitted. □

So we have a congruence which conservatively extends CCS. Note that we are following the formulation of CCS in [19] rather than that of [17]. Processes such as $P + (Q \mid R)$ are not allowed, only guarded choices $\sum_{i \in I} \alpha_i.P_i$. In [17], weak equivalence was not a congruence, whereas in [19] it is.

It would have been more obvious to have the following for the clause for $\lambda \in \mathsf{Vis}$ in Definition 7.1: $P \overset{\lambda}{\Rightarrow}_U P'$ iff there exist P'', P''', U', U'' such that $P \Rightarrow_{U'} P'' \xrightarrow{\lambda}_{U''} P''' \Rightarrow P', U = U' \cup U''$ (i.e. omitting the condition $\mathsf{off}(P'') \subseteq \mathsf{off}(P)$).

We would then have defined weak offer bisimulation and weak offer equivalence based on this more generous definition of $P \overset{\lambda}{\Rightarrow}_U P'$. Let $\overset{\mathsf{off}}{\approx}_{\mathsf{gen}}$ denote this more generous weak offer equivalence. This would give us a strictly larger equivalence, which would fail to be a congruence. As an example, let

$$P \overset{\text{def}}{=} a + \tau.(a + u) \qquad Q \overset{\text{def}}{=} \tau.(a + u) \qquad R \overset{\text{def}}{=} \overline{u}:\overline{a}.b$$

Then $P \overset{\mathsf{off}}{\approx}_{\mathsf{gen}} Q$ but not $P \overset{\mathsf{off}}{\approx} Q$. Moreover $P \mid R \xrightarrow{\tau}_{\overline{u}} \mathbf{0} \mid b$ but $Q \mid R$ cannot perform a sequence of τs and then b, demonstrating that $\overset{\mathsf{off}}{\approx}_{\mathsf{gen}}$ is not a congruence.

Clearly if $P \overset{\mathsf{off}}{\approx} Q$ then $P \overset{\mathsf{off}}{\approx}_{\mathsf{gen}} Q$. In fact, the congruence induced by the more generous version is implied by weak offer equivalence:

Proposition 7.5. *For any P, Q, $P \overset{\mathsf{off}}{\approx} Q$ implies $\mathcal{C}[P] \overset{\mathsf{off}}{\approx}_{\mathsf{gen}} \mathcal{C}[Q]$, for all contexts $\mathcal{C}[\cdot]$.*

Proof. Suppose $P \overset{\mathsf{off}}{\approx} Q$. Let $\mathcal{C}[\cdot]$ be a context. By Theorem 7.4, $\mathcal{C}[P] \overset{\mathsf{off}}{\approx} \mathcal{C}[Q]$. But then $\mathcal{C}[P] \overset{\mathsf{off}}{\approx}_{\mathsf{gen}} \mathcal{C}[Q]$. □

We have not determined whether the converse to Proposition 7.5 holds.

We now turn to the equational theory of weak offer equivalence. In CCS we have the law $P \approx \tau.P$ [19, Theorem 6.15]. However in CPG, $u \overset{\mathsf{off}}{\not\approx} \tau.u$. This is because $\mathsf{off}(u) = \{u\}$ whereas $\mathsf{off}(\tau.u) = \emptyset$. Nonetheless, the usual CCS equivalence laws will still hold for the standard processes $\mathcal{P}_{\mathsf{Std}}$ (recall that for $P \in \mathcal{P}_{\mathsf{Std}}$, $\mathsf{off}(P) = \emptyset$).

Proposition 7.6 (cf. [19, Theorem 6.15]). *The following laws hold*:

$$\tau.P \overset{\mathsf{off}}{\approx} P \quad \text{if } \mathsf{off}(P) = \emptyset \tag{6}$$

$$M + N + \tau.N \overset{\mathsf{off}}{\approx} M + \tau.N \quad \text{if } \mathsf{off}(N) \subseteq \mathsf{off}(M) \tag{7}$$

$$M + \alpha.P + \alpha.(\tau.P + N) \overset{\mathsf{off}}{\approx} M + \alpha.(\tau.P + N) \tag{8}$$

Proof. Straightforward and omitted. □

We stated (6)–(8) because in many situations it is convenient to use conventional CCS reasoning. If we remove the side-conditions concerning offers, these are precisely the three τ-laws of CCS, which, as Milner states, capture the various ways in which weak equivalence allows one to vary the forms of τ transitions.

The next result gives the "intrinsic" τ-laws of CPG:

Proposition 7.7. *The following four laws hold*:

$$M + U:\tau.M \overset{\mathsf{off}}{\approx} M \tag{9}$$

$$M + U:\tau.(N + V:\tau.P) \overset{\mathsf{off}}{\approx} M + U:\tau.(N + V:\tau.P) + (U \cup V):\tau.P \tag{10}$$

If $\mathsf{off}(N + V:\lambda.P) \subseteq \mathsf{off}(M)$:

$$M + U:\tau.(N + V:\lambda.P) \overset{\mathsf{off}}{\approx} M + U:\tau.(N + V:\lambda.P) + (U \cup V):\lambda.P \tag{11}$$

$$M + U:\alpha.P + U:\alpha.(\tau.P + N) \overset{\mathsf{off}}{\approx} M + U:\alpha.(\tau.P + N) \tag{12}$$

Proof. Straightforward and omitted. □

We can derive (7) from (10) and (11). Clearly (8) is a special case of (12), where $U = \emptyset$. Also we can derive:

$$\tau.M \overset{\mathsf{off}}{\approx} M \quad \text{if } \mathsf{off}(M) = \emptyset \tag{13}$$

from (9)–(11). Recall that every process is strongly equivalent to a summation (Proposition 6.6), and so (13) is effectively as strong as (6).

Definition 7.8. Let $\mathcal{A}_{\mathrm{W}}$ be the axioms $\mathcal{A}_{\mathrm{S}}$ (Definition 6.9) together with (9)–(12).

Note that law (3) can be omitted, as it is derivable from (9)–(11).

Before showing the completeness of $\mathcal{A}_{\mathrm{W}}$ for finite processes, we prove a "saturation" lemma:

Lemma 7.9 (cf. [17, Section 7.4, Lemma 16]). *Let P be in standard form and let $U \subseteq V$.*

(1) *If $P \Rightarrow_U P'$ then $\mathcal{A}_{\mathrm{W}} \vdash P = P + V:\tau.P'$.*

(2) *If $P \overset{\lambda}{\Rightarrow}_U P'$ then $\mathcal{A}_{\mathrm{W}} \vdash P = P + V:\lambda.P'$.*

Proof

(1) We use induction on the length of the computation $P \Rightarrow_U P'$. If $P \overset{\mathrm{id}}{=} P'$ then we use (9) to show that $\mathcal{A}_{\mathrm{W}} \vdash P = P + V:\tau.P'$. Otherwise we have $P \overset{\tau}{\rightarrow}_{U'} P'' \Rightarrow_{U''} P'$, with $U' \cup U'' = U$. Now P'' is in standard form, and so by induction hypothesis, $\mathcal{A}_{\mathrm{W}} \vdash P'' = P'' + V:\tau.P'$. We now use (10) to derive $\mathcal{A}_{\mathrm{W}} \vdash P = P + V:\tau.P'$.

(2) Suppose $P \Rightarrow_{U'} P'' \overset{\lambda}{\rightarrow}_{U''} P''' \Rightarrow P'$ with $U = U' \cup U''$ and $\mathsf{off}(P'') \subseteq \mathsf{off}(P)$. By part (1), $\mathcal{A}_{\mathrm{W}} \vdash P''' = P''' + \tau.P'$. We use law (12) to derive $\mathcal{A}_{\mathrm{W}} \vdash P'' = P'' + U'':\lambda.P'$. Again by part (1), $\mathcal{A}_{\mathrm{W}} \vdash P = P + U':\tau.P''$. So $\mathcal{A}_{\mathrm{W}} \vdash P = P + U':\tau.(P'' + U'':\lambda.P')$. Since $\mathsf{off}(P'') \subseteq \mathsf{off}(P)$ and $\lambda \in \mathsf{off}(P'')$, we can use (11) to obtain $\mathcal{A}_{\mathrm{W}} \vdash P = P + U:\lambda.P'$. Finally $\mathcal{A}_{\mathrm{W}} \vdash P = P + V:\lambda.P'$ follows by (3). □

Theorem 7.10. *The axioms $\mathcal{A}_{\mathrm{W}}$ are complete for $\overset{\mathsf{off}}{\approx}$ on finite processes.*

Proof. Suppose that $P \overset{\mathsf{off}}{\approx} Q$. By Lemma 6.12 we can assume P and Q are in standard form. We prove P and Q equal much as in Theorem 6.14: for each summand $U:\alpha.P'$ of P we add a new summand to Q, to form $N + Q$, and for each summand $V:\alpha.Q'$ of Q we add a new summand to P, to form $P + M$.

Suppose $P \overset{\text{id}}{=} \sum U_i : \alpha_i . P_i$. For each i, there are two cases:

If $\alpha_i = \tau$ we have $P \xrightarrow{\tau}_{U_i} P_i$. So $Q \Rightarrow_{V_i} Q'_i$, for some $V_i \subseteq U_i$, with $P_i \overset{\text{off}}{\approx} Q'_i$. By Lemma 7.9, $\mathcal{A}_W \vdash Q = Q + U_i : \tau . Q'_i$. By induction on the total depth of P and Q, we have $\mathcal{A}_W \vdash P_i = Q'_i$ (note that P_i has lower depth than P, even though Q'_i might have the same depth as Q).

If $\alpha_i = \lambda \in \mathsf{Vis}$ we have $P \xrightarrow{\lambda}_{U_i} P_i$. Then there is Q'_i such that $P_i \overset{\text{off}}{\approx} Q'_i$ and $Q \overset{\lambda}{\Rightarrow}_{V_i} Q'_i$, for some $V_i \subseteq U_i$. Again by Lemma 7.9, $\mathcal{A}_W \vdash Q = Q + U_i : \lambda . Q'_i$. By induction on the total depth of P and Q, we have $\mathcal{A}_W \vdash P_i = Q'_i$.

Let $N = \sum U_i : \alpha_i . Q'_i$. Then $\mathcal{A}_W \vdash Q = N + Q$. We get M symmetrically, with $\mathcal{A}_W \vdash P = P + M$. Now $\mathcal{A}_W \vdash P + M = N + Q$ and hence $\mathcal{A}_W \vdash P = Q$ as required. □

When reasoning about recursively specified processes, a common style of reasoning is to show that two processes satisfy the same set of equations, and deduce that the processes are equivalent. We shall use this method in the next section. For it to be sound, we need the following result:

Proposition 7.11 (cf. [19, Theorem 6.19]). *Unique solution of equations. Let $\vec{X}$ be a (possibly infinite) sequence of process variables X_i. Up to $\overset{\text{off}}{\approx}$, there is a unique sequence $\vec{P}$ of processes which satisfy the formal equations*:

$$X_i \overset{\text{off}}{\approx} \sum_{j \in J_i} U_{ij} : \alpha_{ij} . X_{k(ij)}$$

(*where each* $\alpha_{ij} \neq \tau$).

Proof. Much as in [19, Theorem 6.19]. □

The preceding result would also hold for strong offer equivalence $\overset{\text{off}}{\sim}$.

Remark 7.12. Milner [18] provides a complete axiomatisation for finite-state processes with respect to weak equivalence, in the case where recursion is guarded (by visible actions). We conjecture that this can also be carried through for finite-state processes with priority guards, in much the same way. Milner also provides a complete axiomatisation for the case where recursion is unguarded. Here we do not see how to adapt his method to the setting with priority guards. He gives axioms which allow the elimination of recursion variables guarded by τ-actions. These axioms are sound in the priority guard setting, but cannot be adapted in an obvious manner to deal with the case of τ-actions with priority guards.

8. Verifying example systems

In this section we revisit two examples given earlier. Now that we have developed notions of bisimulation and axiom systems, we can verify that the examples do indeed behave as intended.

We start with the interrupt example from Section 1. Recall that we had:

$$P \overset{\text{def}}{=} \mathsf{new}\ \mathsf{mid}, \mathsf{int}_A, \mathsf{int}_B\ (A \mid B \mid I) \qquad A \overset{\text{def}}{=} \mathsf{int}_A : a.\overline{\mathsf{mid}}.A + \mathsf{int}_A$$
$$I \overset{\text{def}}{=} \mathsf{int}.(\overline{\mathsf{int}}_A.\overline{\mathsf{int}}_B + \overline{\mathsf{int}}_B.\overline{\mathsf{int}}_A) \qquad B \overset{\text{def}}{=} \mathsf{int}_B : b.\mathsf{mid}.B + \mathsf{int}_B$$

We want to show $P \overset{\text{off}}{\approx} Q$, where

$$Q \overset{\text{def}}{=} a.Q_1 + b.Q_2 + \mathsf{int} \quad Q_1 \overset{\text{def}}{=} b.Q + \mathsf{int} \quad Q_2 \overset{\text{def}}{=} a.Q + \mathsf{int}$$

Clearly $\mathsf{int}_A, \mathsf{int}_B \in \mathsf{Pri}$. We take $a, b, \mathsf{mid}, \mathsf{int} \in \mathsf{Std}$. This means that $Q \in \mathcal{P}_{\mathsf{Std}}$. We can use laws (4) and (5) to get:

$$P \overset{\text{off}}{\sim} a.P_1 + b.P_2 + \mathsf{int}.P_3$$

$$P_1 \overset{\text{off}}{\sim} b.P_4 + \mathsf{int}.\tau \quad P_2 \overset{\text{off}}{\sim} a.P_4 + \mathsf{int}.\tau \quad P_3 \overset{\text{off}}{\sim} \tau.\tau + \tau.\tau \quad P_4 \overset{\text{off}}{\sim} \tau.P + \mathsf{int}.\tau.P_3$$

where P_1, P_2, P_3, P_4 are various states of P. We can use law (6) to get:

$$P_1 \overset{\text{off}}{\approx} b.P_4 + \mathsf{int} \quad P_2 \overset{\text{off}}{\approx} a.P_4 + \mathsf{int} \quad P_3 \overset{\text{off}}{\approx} \mathbf{0} \quad P_4 \overset{\text{off}}{\approx} \tau.P + \mathsf{int}$$

By law (7) we get $P + \tau.P \overset{\text{off}}{\approx} \tau.P$. Notice that this needs $\mathsf{off}(P) = \emptyset$, i.e. $a, b, \mathsf{int} \notin \mathsf{Pri}$. From this we can deduce $\tau.P + \mathsf{int} \overset{\text{off}}{\approx} \tau.P$ using law (3) and law (7) again, so that $P_4 \overset{\text{off}}{\approx} \tau.P$. Finally:

$$P \overset{\text{off}}{\approx} a.P_1 + b.P_2 + \mathsf{int} \quad P_1 \overset{\text{off}}{\approx} b.P + \mathsf{int} \quad P_2 \overset{\text{off}}{\approx} a.P + \mathsf{int}$$

By Proposition 7.11 we get $P \overset{\text{off}}{\approx} Q$ as we wanted.

Our reasoning was presented equationally, but could equally well have been done using bisimulation. We first unfolded the behaviour of P. Since all prioritised actions were restricted, the system P had no priorities as far as the environment was concerned. We could therefore remove silent actions and simplify using standard techniques of CCS.

We also revisit the memory bench example from Section 2.2. The system Sys was defined as follows:

$$\mathsf{Appl} \overset{\text{def}}{=} \overline{\mathsf{fetch}}_1.\overline{\mathsf{fetch}}_2.\mathsf{Appl}$$

$$\mathsf{Bench}_i \overset{\text{def}}{=} \mathsf{fetch}_i.\mathsf{Bench}_i + \mathsf{fetch}_i : \mathsf{dma}.\mathsf{Bench}_i \quad (i = 1, 2)$$

$$\mathsf{Sys} \overset{\text{def}}{=} \mathsf{new}\ \mathsf{fetch}_1, \mathsf{fetch}_2\ (\mathsf{Appl} \mid \mathsf{Bench}_1 \mid \mathsf{Bench}_2)$$

Using laws (4) and (5) we get

$$\mathsf{Sys} \overset{\text{off}}{\sim} \tau.\mathsf{Sys}' + \mathsf{dma}.\mathsf{Sys} \qquad \mathsf{Sys}' \overset{\text{off}}{\sim} \tau.\mathsf{Sys} + \mathsf{dma}.\mathsf{Sys}'$$

It is then straightforward to check (directly from the definition of $\overset{\text{off}}{\approx}$) that $\mathsf{Sys} \overset{\text{off}}{\approx} P$, where $P \overset{\text{def}}{=} \mathsf{dma}.P$; the bisimulation relation is $\{(\mathsf{Sys}, P), (\mathsf{Sys}', P)\}$.

Remark 8.1. In Sections 1 and 2 we used plain equality when talking about equivalence between CPG processes, for example, $\mathsf{Sys} = P$. This is to be interpreted as $\overset{\text{off}}{\approx}$.

9. Expressiveness

In this section we show that priorities add expressive power to both CCS and the π-calculus [20,19]. As far as we are aware, this has not been previously shown for any notion of priority in process algebra.

Bougé [4] showed in the context of CSP [12] that the problem of leader election in symmetric networks of processes can be useful in measuring expressiveness. Palamidessi [23] used electoral systems to show that the π-calculus with mixed choice is more powerful than CCS. We have been inspired by this work, and also the work of Ene and Muntean [9], where it is shown that a broadcast version of the π-calculus cannot be encoded under certain conditions into the π-calculus.

The *leader election problem* is a well-known problem in distributed computing. In order to restart a network of processes after a reconfiguration or a crash, in the absence of a central server, the processes must elect a leader process from among themselves. This process may be thought of as the "winner" in the election, with the other processes being "losers". An algorithm to do this is called an *electoral system*. An electoral system is *symmetric* if it works for a network where all the processes are programmed in the same way. Such a system clearly has to find a way to break symmetry in order to choose a leader.

The general idea for our expressiveness results is that if a language has electoral systems then it has expressive strength. We look for conditions on encodings between languages which ensure that electoral systems are preserved. If we find one language which does have electoral systems and another which does not, then the first language cannot be encoded into the second (under our conditions).

We first discuss electoral systems for CPG (Section 9.1). We next recall the π-calculus (Section 9.2). We then prove the nonexistence of an encoding from CPG to the π-calculus under certain conditions (Section 9.3), and finally prove the nonexistence of an encoding from the π-calculus to CPG under certain conditions (Section 9.4).

9.1. Electoral systems for CPG

In this section we show (Theorem 9.10) that CPG without choice has a symmetric electoral system of every finite size k, which is defined in a uniform fashion, i.e. essentially the same process will do for every value of k.

We start by defining electoral systems. The definitions we give below are much as in [25,26], to which we refer the reader for further discussion.

We need a notion of barb (observation):

Definition 9.1. A CPG process P *exhibits barb* λ, written as $P \downarrow \lambda$, iff $P \equiv \mathsf{new}\,\vec{a}\ ((S{:}\lambda.Q + M) \mid R)$ for some $\vec{a}$, S, Q, M and R, with $\mathsf{name}(\lambda) \notin \vec{a}$.

We use unconditional reactions (Section 4) rather than labelled transitions in our definition of computation, as this facilitates comparison between different process calculi, in our case CPG and the π-calculus.

Definition 9.2. Let P be a CPG process. A *computation* $\mathcal{C}$ of P is a (finite or infinite) sequence $P = P_0 \to P_1 \to \cdots$. It is *maximal* if it cannot be extended.

We assume that $\mathcal{N}$ includes a set of *observables* $\mathsf{Obs} = \{\omega_i : i \in \mathbb{N}\}$ such that for all i, j we have $\omega_i \neq \omega_j$ if $i \neq j$.

Definition 9.3. Let $\mathcal{C}$ be a computation $P_0 \to \cdots \to P_i \to \cdots$. The *observables* of $\mathcal{C}$ are $\mathsf{Obs}(\mathcal{C}) = \{\omega \in \mathsf{Obs} : \exists i\ \ P_i \downarrow \omega\}$.

Networks are collections of processes running in parallel:

Definition 9.4 (*cf.* [23]). A *network* Net of size k is a pair $(\vec{a}, \langle P_0, \ldots, P_{k-1}\rangle)$, where $\vec{a}$ is a finite sequence of names and $P_0, \ldots, P_{k-1}$ are processes. The *process interpretation* of Net is the process $\mathsf{new}\,\vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$.

Networks inherit a notion of computation from their process interpretation in an obvious fashion. In what follows we shall write networks in their process interpretation as restricted parallel compositions.

Definition 9.5. A *permutation* is a bijection $\sigma : \mathcal{N} \cup \mathcal{U} \to \mathcal{N} \cup \mathcal{U}$ such that σ preserves the distinction between observable and non-observable names, i.e. $a \in \mathsf{Obs}$ iff $\sigma(a) \in \mathsf{Obs}$. Any permutation σ gives rise to a mapping on processes, where $\sigma(P)$ is the same as P, except that any name a of P is changed to $\sigma(a)$ in $\sigma(P)$.

A permutation σ induces a bijection $\hat{\sigma} : \mathbb{N} \to \mathbb{N}$ defined as follows: $\hat{\sigma}(i) = j$ where $\sigma(\omega_i) = \omega_j$. Thus for all $i \in \mathbb{N}$, $\sigma(\omega_i) = \omega_{\hat{\sigma}(i)}$. We use $\hat{\sigma}$ to permute the indices of processes in a network.

Definition 9.6. Let $\mathsf{Net} = \mathsf{new}\,\vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$ be a network of size k. An *automorphism* on Net is a permutation σ such that (1) $\hat{\sigma}$ restricted to $\{0, \ldots, k-1\}$ is a bijection, and (2) σ preserves the distinction between free and bound names, i.e. $a \in \vec{a}$ iff $\sigma(a) \in \vec{a}$. If $\hat{\sigma}$ restricted to $\{0, \ldots, k-1\}$ is not the identity we say σ is *non-trivial*.

Definition 9.7. Let σ be an automorphism on a network of size k. For any $i \in \{0, \ldots, k-1\}$ the *orbit* $\mathcal{O}_{\hat{\sigma}}(i)$ generated by $\hat{\sigma}$ is defined as follows:

$$\mathcal{O}_{\hat{\sigma}}(i) = \{i, \hat{\sigma}(i), \hat{\sigma}^2(i), \ldots, \hat{\sigma}^{h-1}(i)\}$$

where $\hat{\sigma}^j$ represents the composition of $\hat{\sigma}$ with itself j times, and h is least such that $\hat{\sigma}^h(i) = i$. If every orbit has the same size then σ is *well-balanced*.

Definition 9.8. Let $\mathsf{Net} = \mathsf{new}\,\vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$ be a network of size k and let σ be an automorphism on it. We say that Net is *symmetric with respect to* σ iff for each $i = 0, \ldots, k-1$ we have $P_{\hat{\sigma}(i)} = \sigma(P_i)$. We say that Net is *symmetric* if it is symmetric with respect to some automorphism with a single orbit (which must have size k).

An electoral system is a network where every possible maximal computation elects a unique leader:

Definition 9.9. A network Net of size k is an *electoral system* if for every maximal computation $\mathcal{C}$ of Net there exists an $i < k$ such that $\mathsf{Obs}(\mathcal{C}) = \{\omega_i\}$.

The intuition behind the following theorem is that priorities give us something of the power of one-many (broadcast) communication, in that a single process can simultaneously interrupt several other processes. By contrast, CCS and π-calculus communication are always one-one.

Theorem 9.10. *For every $k \geq 1$ there is a symmetric network of size k in CPG without choice which is an electoral system.*

Proof. Let $R_i \stackrel{\text{def}}{=} u{:}\overline{a}.(\omega_i \mid \overline{u}) \mid a$. Let $\mathsf{Net} \stackrel{\text{def}}{=} \mathsf{new}\ u\ (R_0 \mid \cdots R_{k-1})$. Clearly Net is symmetric. We show that Net is an electoral system. Suppose that $\mathsf{Net} \rightarrow P$. This reaction must involve the $u{:}\overline{a}$ of R_i and the a of R_j, for some $i,\ j \leq k-1$. Plainly $P \downarrow \omega_i$. Also no further reactions are possible, now that the $\overline{u}$ of R_i is exposed. So every computation has a unique winner. □

Note that the solution is uniform, in that the R_i do not depend on the size k of the network; the R_i do not "know" k. We need the restriction on u in order that reactions are unconditional.

The uniform electoral system in the proof of Theorem 9.10 can be translated into CW's approach with R_i becoming $(u.\overline{u} + a.(\omega_i \mid \overline{u})) \mid \overline{a}$. Of course even though this does not contain choice, it does use priority choice. The "broadcast" flavour is rather less in the CW version, since the $\overline{u}$ produced by the winning process is consumed and regenerated by the losing processes, rather than simply inhibiting them from further action as in the CPG version. We can also create a CLN version of the CW uniform electoral system, but here we do seem to need choice.

We have adopted a definition of electoral system where the losing processes do not need to announce the winner—they simply lose and do nothing. If we wanted losing processes also to report the winner (as in the formulation of electoral systems in [23]) we could still get a uniform solution if we added value-passing to CPG. Then the processes could share a channel c on which the winner could repeatedly (actually k times, but the winner would not need to have this in its program) send its index i for all the processes to receive and then output on a common output channel o.

We shall build on Theorem 9.10 in showing the nonexistence of encodings from CPG to the π-calculus (Section 9.3).

9.2. The π-calculus

We follow the version of the π-calculus given in [19]. We let a, b range over names, and λ range over names a or co-names $\overline{a}$. Processes are given by the following syntax:

$$P, Q ::= \sum_{i \in I} \pi_i.P_i \ \mid\ P \mid Q \ \mid\ \mathsf{new}\ a\ P \ \mid\ !\,P$$

where I is finite, the *prefixes* are given by

$$\pi ::= a(b) \ \mid\ \overline{a}\langle b\rangle \ \mid\ \tau$$

and $!\,P$ is replication. We refer the reader to [19] for the definition of reaction $P \rightarrow Q$. Computations and networks are as for CPG. It is helpful to define barbs in the π-calculus by structural induction, rather than in the style of Definition 9.1:

Definition 9.11. *Barbs* are defined by structural induction on π-calculus processes as follows:

$(\sum_{i\in I} \pi_i.P_i) \downarrow a$ iff $\exists i \in I\ \exists b.\pi_i = a(b)$
$(\sum_{i\in I} \pi_i.P_i) \downarrow \overline{a}$ iff $\exists i \in I\ \exists b.\pi_i = \overline{a}\langle b\rangle$
$(P \mid Q) \downarrow \lambda$ iff $P \downarrow \lambda$ or $Q \downarrow \lambda$
$(\mathsf{new}\ a\ P) \downarrow \lambda$ iff $P \downarrow \lambda$ and $\mathsf{name}(\lambda) \neq a$
$!\,P \downarrow \lambda$ iff $P \downarrow \lambda$

When proving that CPG cannot be encoded in the π-calculus (Theorem 9.16), we shall require some properties of the π-calculus, which we state in the following two lemmas:

Lemma 9.12. *Let P, Q be π-calculus processes. If* $\mathsf{new}\ a\ P \to Q$ *then there exists Q' such that $P \to Q'$.*

Proof (*Sketch*). If we take a derivation of $\mathsf{new}\ a\ P \to Q$ and delete all the relevant occurrences of $\mathsf{new}\ a$, then we have a derivation of $P \to Q'$ for some Q'. We omit the details. □

Lemma 9.13

(1) *For any π-calculus processes P_1, P_2, whenever P_i has a maximal computation with observables O_i ($i = 1, 2$) then $P_1 \mid P_2$ has a maximal computation with observables O such that $O_1 \cup O_2 \subseteq O$.*
(2) *For any π-calculus process P, any $O \subseteq \mathsf{Obs}$ and any $a \notin O$, if P has a maximal computation with observables O then* $\mathsf{new}\ a\ P$ *has a maximal computation with observables O.*

Proof

(1) Suppose that we have maximal computations $\mathcal{C}_1 = P_1 \to P_1^1 \to P_1^2 \to \cdots$ and $\mathcal{C}_2 = P_2 \to P_2^1 \to \cdots$ with observables O_1, O_2 respectively. We can interleave $\mathcal{C}_1$ and $\mathcal{C}_2$ to get a new computation $\mathcal{C} = P_1 \mid P_2 \to P_1^1 \mid P_2 \to P_1^1 \mid P_2^1 \to P_1^2 \mid P_2^1 \to \cdots$. This will have observables $O_1 \cup O_2$. If $\mathcal{C}$ is maximal then we are done. Otherwise we extend $\mathcal{C}$ to a maximal computation $\mathcal{C}'$, with observables O such that $O_1 \cup O_2 \subseteq O$.
(2) Suppose we have a maximal computation $\mathcal{C} = P \to P_1 \to \cdots$ such that $\mathsf{Obs}(\mathcal{C}) = O$. Suppose that $a \notin O$. Then we have a computation $\mathcal{C}' = \mathsf{new}\ a\ P \to \mathsf{new}\ a\ P_1 \to \cdots$. If $\mathcal{C}$ is infinite, then so is $\mathcal{C}'$, which is therefore maximal. If $\mathcal{C}$ is finite, then $\mathcal{C}'$ is also maximal, using Lemma 9.12.
Now we must show that $\mathsf{Obs}(\mathcal{C}') = O$. But $\mathsf{Obs}(\mathcal{C}') = \{\omega_j : \exists i.(\mathsf{new}\ a\ P_i) \downarrow \omega_j\}$. Also for any i, j, $(\mathsf{new}\ a\ P_i) \downarrow \omega_j$ iff $P_i \downarrow \omega_j$, since $a \notin O$. Hence $\mathsf{Obs}(\mathcal{C}') = \{\omega_j : \exists i.P_i \downarrow \omega_j\} = O$ as required. □

9.3. Nonexistence of encodings from CPG to the π-calculus

In this section we prove (Theorem 9.16) that there is no encoding from CPG into the π-calculus satisfying certain conditions. This will show that priorities do indeed add expressiveness.

Our conditions are defined for *process languages*, by which we mean languages which, following the π-calculus paradigm, are equipped with a notion of name (and free name), a parallel composition operator, a name-binding restriction operator, a reduction relation and a notion of barb.

We recall the following from [25] (building on [23]):

Definition 9.14. Let L, L' be process languages. An encoding $[\![-]\!] : L \to L'$ is

(1) *distribution-preserving* if for all processes P, Q of L, $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$;
(2) *permutation-preserving* if for any permutation of names σ in L there exists a permutation θ in L' such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ and the permutations are compatible on observables, in that for all $i \in \mathbb{N}$ we have $\sigma(\omega_i) = \theta(\omega_i)$;
(3) *observation-respecting* if for any P in L,
   (a) for every maximal computation $\mathcal{C}$ of P there exists a maximal computation $\mathcal{C}'$ of $[\![P]\!]$ such that $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}')$;
   (b) for every maximal computation $\mathcal{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathcal{C}'$ of P such that $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}')$.

An encoding which preserves distribution and permutation is *uniform*.

We need the following further condition, which is satisfied by many encodings in the literature:

Definition 9.15 (*cf.* [9]). Let L, L' be process languages. An encoding $[\![-]\!] : L \to L'$ is *restriction-preserving* if for all processes P of L and names a, there are names $\vec{b}$ such that $[\![\mathsf{new}\ a\ P]\!] = \mathsf{new}\ \vec{b}\ [\![P]\!]$, and furthermore, if $a \notin \mathsf{Obs}$ then $\vec{b} \notin \mathsf{Obs}$.

Note that in Definition 9.15 the names $\vec{b}$ can vary depending on the process P, and not just on the name a.

The following theorem is inspired by Ene and Muntean's result [9] that the broadcast π-calculus cannot be encoded in the π-calculus.

Theorem 9.16. *There is no distribution-preserving, restriction-preserving and observation-respecting encoding of CPG into the π-calculus.*

Proof. Suppose that $[\![-]\!]$ is a distribution-preserving, restriction-preserving and observation-respecting encoding of CPG into the π-calculus. For $i = 0, 1$ let $R_i \stackrel{\text{def}}{=} u{:}\overline{a}.(\omega_i \mid \overline{u}) \mid a$, as in the proof of Theorem 9.10. Then $\mathsf{new}\ u\ R_0$ is an electoral system of size one. It is easy to see that $[\![\mathsf{new}\ u\ R_0]\!]$ is also an electoral system, using the observation-respecting condition. But $[\![\mathsf{new}\ u\ R_0]\!] = \mathsf{new}\ \vec{a}\ [\![R_0]\!]$ for some names $\vec{a}$. Note that $\vec{a} \notin \mathsf{Obs}$, since $u \notin \mathsf{Obs}$. Clearly $[\![R_0]\!]$ is also an electoral system: for every maximal computation $\mathcal{C}$ with observables O we have a corresponding maximal computation of $\mathsf{new}\ \vec{a}\ [\![R_0]\!]$ with observables O by Lemma 9.13(2), so that we must have $O = \{\omega_0\}$.

Similarly we can show that every maximal computation of $[\![R_1]\!]$ has observables $\{\omega_1\}$.

From the proof of Theorem 9.10, $\mathsf{Net} \stackrel{\text{def}}{=} \mathsf{new}\ u\ (R_0 \mid R_1)$ is an electoral system of size two. Then $[\![\mathsf{Net}]\!]$ is also an electoral system of size two. Now $[\![\mathsf{Net}]\!] = [\![\mathsf{new}\ u\ (R_0 \mid R_1)]\!] = \mathsf{new}\ \vec{b}\ ([\![R_0]\!] \mid [\![R_1]\!])$. By Lemma 9.13(1), there is a maximal computation of $[\![R_0]\!] \mid [\![R_1]\!]$ with observables O such that $\{\omega_0, \omega_1\} \subseteq O$. But then there is a maximal computation of $\mathsf{new}\ \vec{b}\ ([\![R_0]\!] \mid [\![R_1]\!])$ with observables O (by Lemma 9.13(2)), which contradicts $[\![\mathsf{Net}]\!]$ being an electoral system. □

Since CCS can be encoded in the π-calculus, it follows that CPG has greater expressive power than CCS. It also follows that we can add expressive power to the π-calculus by adding priority guards.

Note that Lemma 9.13 is equally true for any process calculus which treats parallel composition and restriction in the same way as the π-calculus, and has a notion of barb. An example is Mobile Ambients [6]. Theorem 9.16 would therefore remain true if we replaced the π-calculus by Mobile Ambients.

9.4. Nonexistence of encodings from the π-calculus to CPG

In this section we show the converse to Theorem 9.16: there is no encoding from the π-calculus into CPG satisfying certain conditions (Theorem 9.27). This was shown by Palamidessi [23] in the case of CCS rather than CPG. The idea is that π-calculus networks can acquire new links, while CCS networks cannot. The separation result comes from showing that, for certain non-fully-connected networks, symmetric electoral systems exist in the π-calculus, but not in CCS. Essentially the same is true for CPG, but the added power of priorities mean that Palamidessi's negative result for CCS does not hold as it stands for CPG.

First we recall some definitions from [26] which relate to connectivity in networks. They apply equally to CPG, CCS and the π-calculus.

Definition 9.17. Two processes P and Q are *independent* if they do not share any free names: $\mathsf{fn}(P) \cap \mathsf{fn}(Q) = \emptyset$.

Definition 9.18. Let σ be an automorphism on a network $\mathsf{Net} = \mathsf{new}\ \vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$. Then Net is *independent* with respect to σ if every orbit forms an independent set, in the sense that if $i, j < k$ are in the same orbit of $\hat{\sigma}$ with $i \neq j$, then P_i and P_j are independent.

Definition 9.19. An encoding is *independence-preserving* if for any processes P, Q, if P and Q are independent then $[\![P]\!]$ and $[\![Q]\!]$ are also independent.

Definition 9.20. A *ring* is a network $\mathsf{Net} = \mathsf{new}\ \vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$ which has a single-orbit automorphism σ such that for all $i, j < k$, if $\mathsf{fn}(P_i) \cap \mathsf{fn}(P_j) \neq \emptyset$ then one of $i = j$, $\hat{\sigma}(i) = j$ or $\hat{\sigma}(j) = i$ must hold. A ring is *symmetric* if it is symmetric with respect to such an automorphism σ.

Rings are mapped to rings by encodings which preserve independence among other conditions:

Lemma 9.21 ([26]). *Suppose $[\![-]\!] : L \to L'$ is a uniform, observation-respecting and independence-preserving encoding. Suppose that* Net *is a symmetric ring of size $k \geq 1$ with no globally-bound names which is an electoral system. Then $[\![\mathsf{Net}]\!]$ is also a symmetric ring of size k with no globally-bound names which is an electoral system.*

The next result is essentially Theorem 6.1 of [23]:

Theorem 9.22. *If* Net *is a CCS network which is symmetric and independent with respect to a non-trivial well-balanced automorphism σ, then* Net *is not an electoral system.*

Palamidessi points out the following application of Theorem 9.22: if we have a ring $\mathsf{Net} = \mathsf{new}\,\vec{a}\ (P_0 \mid \cdots \mid P_3)$ of size four, which is symmetric with respect to an automorphism σ satisfying $\hat{\sigma}(i) = i + 2$ (using addition modulo 4), then σ is non-trivial, well-balanced and has independent orbits, and we deduce that Net is not an electoral system.

Palamidessi uses this to show the nonexistence of an encoding from the π-calculus to CCS. She defines a symmetric ring of size four in the π-calculus which is an electoral system. This is mapped by any uniform, observation-respecting and independence-preserving encoding to a symmetric ring of size four in CCS which is an electoral system, by Lemma 9.21. But this contradicts Theorem 9.22.

Theorem 9.22 does not hold for CPG, since the effects of a communication between two processes P_i and P_j extend beyond those two processes, because it may uncover priority actions in P_i or P_j which inhibit other processes in the network.

Proposition 9.23. *There is a symmetric ring of size four in CPG without choice which is an electoral system.*

Proof. For $i = 0, \ldots, 3$, let

$$P_i \stackrel{\text{def}}{=} u_i : \overline{a}_{i+1}.(\omega_i \mid \overline{u}_i) \mid \{u_i, u_{i+1}\} : a_i.\overline{u}_i$$

where addition is modulo 4. Let $\mathsf{Net} = \mathsf{new}\,\vec{u}, \vec{a}\ (P_0 \mid P_1 \mid P_2 \mid P_3)$. Note that all the names of P_i are indexed with i or $i + 1$. So therefore P_i and P_{i+2} are independent for $i < 4$, and Net is a symmetric ring of size four.

We check that it is an electoral system. Suppose without loss of generality that the first reaction is between P_0 and P_1. We have

$$\begin{aligned}
&\mathsf{new}\,\vec{u}, \vec{a}\ (P_0 \mid P_1 \mid P_2 \mid P_3)\\
&\quad \to \mathsf{new}\,\vec{u}, \vec{a}\ ((\omega_0 \mid \overline{u}_0) \mid \{u_0, u_1\} : a_0.\overline{u}_0\\
&\quad \mid u_1 : \overline{a}_2.(\omega_1 \mid \overline{u}_1) \mid \overline{u}_1\\
&\quad \mid u_2 : \overline{a}_3.(\omega_2 \mid \overline{u}_2) \mid \{u_2, u_3\} : a_2.\overline{u}_2\\
&\quad \mid u_3 : \overline{a}_0.(\omega_3 \mid \overline{u}_3) \mid \{u_3, u_0\} : a_3.\overline{u}_3)
\end{aligned}$$

Plainly 0 has won, since there is an ω_0 barb. Also it can be checked that no further reaction is possible. Hence Net is indeed an electoral system. □

In view of Proposition 9.23 we must change the conditions in Theorem 9.22 to get a version that holds for CPG.

We can associate a simple undirected graph with a network $\mathsf{Net} = \mathsf{new}\,\vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$ by letting the nodes be $0, \ldots, k-1$ and joining i to j iff P_i and P_j are not independent. We define the *distance* $d(i, j)$ between i and j to be the length of the shortest path from i to j in the graph (if there is no path from i to j we let $d(i, j) = \infty$). The independent orbit condition in Theorem 9.22 can be seen as requiring that if i and j are in the same orbit then $i = j$ or $d(i, j) \geq 2$. To get the analogue of Theorem 9.22 for CPG we increase the minimum distance between members of the same orbit:

Theorem 9.24. *Let* Net *be a CPG network which is symmetric with respect to a non-trivial well-balanced automorphism σ such that whenever i and j are in the same orbit of σ then $i = j$ or $d(i, j) \geq 3$. Then* Net *is not an electoral system.*

Proof (*Sketch*). The proof is based on the method Palamidessi used to show Theorem 9.22. Let $\mathsf{Net} = \mathsf{new}\,\vec{a}\ (P_0 \mid \cdots \mid P_{k-1})$. Let the size of each orbit be m (all orbits have the same size, since σ is well-balanced). Note that $m > 1$, since σ is non-trivial. We construct a maximal computation which preserves symmetry. If no winner is declared in the computation then Net is not an electoral system. If on the other hand a winner is declared, then by symmetry

all other processes in the same orbit can also be declared winners, which again means that Net is not an electoral system.

Suppose that we have reached state $\mathsf{Net}' = \mathsf{new}\,\vec{a}'\ (P'_0 \mid \cdots \mid P'_{k-1})$ in the computation we are constructing, where Net' is symmetric with respect to σ' and Net' has no new connections between processes compared with Net. Here σ' is the same as σ except that we have adjusted to maintain symmetry for any new restricted names introduced during the computation. If no reaction is enabled then the computation is maximal and we are done. So suppose that some reaction is enabled. There are two cases.

(1) If the reaction involves a single process, $P'_i \rightarrow_U P''_i$, then we perform it and restore symmetry by performing the $m-1$ corresponding reactions round the orbit of i. No new connections are created in the associated graph, by Lemma 4.9. Unlike in the CCS case, the ability to perform the reaction round the orbit of i depends on the processes in the orbit being independent. Otherwise the $P'_i \rightarrow_U P''_i$ reaction might cause a priority action u to be offered by P''_i, which could inhibit a process in i's orbit which shared the name u. Having performed the m reactions, we modify σ' to σ'' as necessary to take into account any new restricted names which may have been introduced.

(2) If the reaction is a communication between two different processes, $P'_i \mid P'_j \rightarrow_U P''_i \mid P''_j$, then i and j must be in different orbits. We perform the reaction and restore symmetry by performing the $m-1$ corresponding reactions around the orbits of both i and j. Again no new connections are created in the associated graph, using Lemma 5.6. The crucial issue is whether any of the reactions can inhibit each other by causing new priority actions to be offered. Consider a node $i' < k$ with $i' \neq i, j$. $P'_{i'}$ can only be affected by the reaction between P'_i and P'_j if i' is connected to either i or j. Suppose that i' is connected to i. Then i' cannot be in the orbit of i or of j, since $d(i', i) = 1$ and $d(i', j) \leq 2$. Similarly if i' is connected to j then i' cannot be in the orbit of i or of j. It follows that the m reactions cannot inhibit each other. Again we modify σ' to σ'' as necessary to take into account any new restricted names introduced during the m reactions. □

Corollary 9.25. *For any composite (non-prime) $k \geq 6$, if* Net *is a symmetric ring of size k in CPG then* Net *is not an electoral system.*

Proof. Let k be composite with $k \geq 6$. Then $k = mn$, where $m \geq 3$. Suppose that Net is a symmetric ring of size k which is symmetric with respect to σ. Then Net is also symmetric with respect to σ^m (σ composed with itself $m-1$ times), which is non-trivial, well-balanced and such that whenever i and j are in the same orbit of σ then $i = j$ or $d(i, j) \geq m \geq 3$. By Theorem 9.24, Net is not an electoral system. □

Matters are different in the π-calculus without restriction:

Proposition 9.26. *For any $k \geq 1$, there is a symmetric ring of size k in the π-calculus without restriction which is an electoral system.*

Proof. (Sketch) Palamidessi showed this for a ring of four with the use of restriction [23, Proposition 5.1]. Her method can be easily adapted to rings of any finite size. Also her method can be adapted to dispense with restriction. We omit the details. Note that, since we are considering unlabelled transitions (reactions), we do not need restriction in order to prevent unwanted labelled transitions. □

We can now prove our main result:

Theorem 9.27. *There is no uniform, observation-respecting and independence-preserving encoding of the π-calculus into CPG.*

Proof. By Proposition 9.26, Lemma 9.21 and Corollary 9.25. □

10. Conclusions

We have introduced priority guards into CCS to form the language CPG. We have defined both strong and weak bisimulation equivalences and seen that they are conservative over the CCS equivalences, and that they are congruences.

We have given complete equational laws for finite CPG in both the strong and weak cases. Conservation over CCS has the consequence that in verifying CPG systems we can often use standard CCS reasoning, as long as we take some care with actions in the set of priority actions Pri.

CPG overcomes the asymmetry between inputs and outputs present both in Camilleri and Winskel's calculus and in the corresponding calculus of Cleaveland, Lüttgen and Natarajan.

Finally, using leader election problems, we have seen that priority guards add expressiveness to both CCS and the π-calculus.

An obvious direction for further work is to work out the implications of adding priority guards to the π-calculus. This has been tackled in [3].

Acknowledgments

We wish to thank Philippa Gardner, Rajagopal Nagarajan, Catuscia Palamidessi, Andrew Phillips, Irek Ulidowski, Maria Grazia Vigliotti, Nobuko Yoshida and the anonymous Concur 2001 referees for helpful discussions and suggestions. We particularly thank this journal's referees for many improvements. This research was partially funded by EPSRC grant GR/K54663.

References

[1] J.C.M. Baeten, J. Bergstra, J.W. Klop, Syntax and defining equations for an interrupt mechanism in process algebra, Fundam. Inform., 9 (1986) 127–168.
[2] J. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inform. and Control 60 (1–3) (1984) 109–137.
[3] A. Bloch, M.V. Frederiksen, B. Haagensen, A π-calculus with prioritized actions with applications, DAT5 Project Report, Group B1-215B, supervisor H. Hüttel, Department of Computer Science, Aalborg University, 2004.
[4] L. Bougé, On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes, Acta Inform. 25 (1988) 179–201.
[5] J. Camilleri, G. Winskel, CCS with priority choice, Inform. and Comput. 116 (1) (1995) 26–37.
[6] L. Cardelli, A.D. Gordon, Mobile ambients, Theoret. Comput. Sci. 240 (1) (2000) 177–213.
[7] R. Cleaveland, M.C.B. Hennessy, Priorities in process algebra, Inform. and Comput. 87 (1/2) (1990) 58–77.
[8] R. Cleaveland, G. Lüttgen, V. Natarajan, Priority in process algebra, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 2001, pp. 391–424 (Chapter 12).
[9] C. Ene, T. Muntean, Expressiveness of point-to-point versus broadcast communications, FCT '99, Lecture Notes in Computer Science, vol. 1684, Springer-Verlag, 1999, pp. 258–268.
[10] C.J. Fidge, A formal definition of priority in CSP, ACM Trans. Programming Languages Syst. 15 (4) (1993) 681–705.
[11] H. Hansson, F. Orava, A process calculus with incomparable priorities, Proceedings of the North American Process Algebra Workshop, Stony Brook, New York, Workshops in Computing, Springer-Verlag, 1992, pp. 43–64.
[12] C.A.R. Hoare, Communicating sequential processes, Commun. Assoc. Comput. Mach. 21 (8) (1978) 666–677.
[13] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
[14] A. Jeffrey, A typed, prioritized process algebra, Technical Report 13/93, Department of Computer Science, University of Sussex, 1993.
[15] C.-T. Jensen, Prioritized and Independent Actions in Distributed Computer Systems, Ph.D. Thesis, Aarhus University, 1994.
[16] R. Milner, A complete inference system for a class of regular behaviours, J. Comput. Syst. Sci. 28 (1984) 439–466.
[17] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[18] R. Milner, A complete axiomatisation for observational congruence of finite-state behaviours, Inform. and Comput. 81 (1989) 227–247.
[19] R. Milner, Communicating and Mobile Systems: the π-calculus, Cambridge University Press, 1999.
[20] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, Inform. and Comput. 100 (1992) 1–77.
[21] V. Natarajan, L. Christoff, I. Christoff, R. Cleaveland, Priorities and abstraction in process algebra, in: P.S. Thiagarajan (Ed.), Foundations of Software Technology and Theoretical Computer Science, 14th Conference, Lecture Notes in Computer Science, vol. 880, Springer-Verlag, 1994, pp. 217–230.
[22] U. Nestmann, B. Pierce, Decoding choice encodings, Inform. and Comput. 163 (1) (2000) 1–59.
[23] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous π-calculi, Math. Structures Comput. Sci. 13 (5) (2003) 685–719.
[24] I.C.C. Phillips, CCS with priority guards, Proceedings of 12th International Conference on Concurrency Theory, CONCUR 2001, Lecture Notes in Computer Science, vol. 2154, Springer-Verlag, 2001, pp. 305–320.
[25] I.C.C. Phillips, M.G. Vigliotti, Electoral systems in ambient calculi, Proceedings of 7th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2004, Lecture Notes in Computer Science, vol. 2987, Springer-Verlag, 2004, pp. 408–422.
[26] I.C.C. Phillips, M.G. Vigliotti, Leader election in rings of ambient processes, Theoret. Comput. Sci. 356 (3) (2006) 468–494.
[27] K.V.S. Prasad, A calculus of broadcasting systems, Sci. Comput. Programming 25 (2–3) (1995) 285–327.