CCS with Priority Guards

Iain Phillips*

Department of Computing, Imperial College, London
iccp@doc.ic.ac.uk

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 show that priority adds expressiveness to both CCS and the π-calculus.

1 Introduction

It has long been recognised that standard process algebra [13,9,2] 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 [1,10], CCS [4,3] and CSP [8,7]. 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 [4], or else they impose a distinction between outputs and inputs, whereby prioritised choice is only made on inputs [3,5]. This goes against the spirit of CCS, where inputs and outputs are treated symmetrically, and 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 show that priority adds expressiveness to both CCS and the π-calculus.

We start with the idea of priority. We assume some familiarity with CCS notation [13]. 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 $\bar{a}$ or $\bar{b}$. By "environment" we

* Partially funded by EPSRC grant GR/K54663

K.G. Larsen and M. Nielsen (Eds.): CONCUR 2001, LNCS 2154, pp. 305–320, 2001.

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 is conditional on the environment not offering actions in $\bar{U}$, 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 [4] 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 [3]. In their notation the example becomes $a.P +\!\!\rangle b.Q$. 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 with these two existing ones in Sect. 2.

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. As a simple example, $a.P + \bar{b}.Q$ can be encoded with priority guards as $\mathrm{new}\, c(c:a.(P|\bar{c})|c:\bar{b}.(Q|\bar{c}))$ (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

$$\mathrm{new}\, c\, ((c +\!\!\rangle a.(P|\bar{c})|(c +\!\!\rangle b.(Q|\bar{c})))$$

but of course here we are exchanging one form of choice for another. We shall return to this encoding of summation in Sect. 6.

To end this section, we give an example, involving 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 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.

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

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. We consider this example more precisely in Sect. 8.

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. Thus priority can be encapsulated locally, which we regard as an important feature when programming larger systems, where different priority frameworks may be in use in different subsystems.

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

2 Comparison with Related Work

We refer the reader to [5] 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 [3] (referred to as CW for short) and Cleaveland, Lüttgen and Natarajan [5] (CLN for short).

2.1 Camilleri and Winskel (CW)

As we have seen, CW's CCS with a prioritised choice operator $P +\!\rangle Q$ 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 [11]).

As we saw in the Introduction, reasoning about priority has to be parametrised on the environment. The CW transition relation is parametrised

on a set of output actions R. Thus $\vdash_R P \xrightarrow{\alpha} P'$ 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, $\vdash_R a{+\!\rangle}b \xrightarrow{a} 0$ (any R), while $\vdash_R a{+\!\rangle}b \xrightarrow{b} 0$ provided $\bar{a} \notin R$.

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

Note that the CW syntax shows the environment only in the prioritised choice $a.P{+\!\rangle}b.Q$, 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 new $a, b\ ((a{+\!\rangle}\bar{b})|(b{+\!\rangle}\bar{a}))$. 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 at the end of Sect. 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.

2.2 Cleaveland, Lüttgen, and Natarajan (CLN)

In CLN's basic approach [5], which is derived from earlier work of Cleaveland and Hennessy [4], actions have priority levels. Mostly they 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 [15]), 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 [3].

We referred in the Introduction to the desirability of encapsulating priorities locally. This encapsulation is present in Camilleri and Winskel's calculus (and in

our own), but not in Cleaveland and Hennessy's, since a high priority $\underline{\tau}$ is treated differently from a standard τ. However, the development in [5] goes beyond the basic Cleaveland and Hennessy calculus to consider distributed priorities, where preemption is decided locally rather than globally. Consideration is also given to 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

$$(a + \underline{b})|(\bar{b} + \underline{c})|\underline{\bar{c}}$$

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.

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.

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 [13, Part I] there is a set of *names* $\mathcal{N}$ and a disjoint set of *co-names* $\bar{\mathcal{N}}$, together with a single silent action τ. To these standard names $\mathcal{N}$ we shall add a new disjoint set of names $\mathcal{U}$ and a dual set $\bar{\mathcal{U}}$. 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 there is a context in which the two sides behave differently. Indeed, $a|\bar{a}:b$ cannot perform b (since, as we shall see, b is preempted by the offer of a), whereas $\tau.a|\bar{a}:b$ can perform b initially, as a is not offered until τ has occurred. 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.

[1] We are following the formulation of CCS in [13] rather than that of [12]. Processes such as $P + (Q|R)$ are not allowed, only guarded choices $\sum \alpha_i.P_i$.

We define $\mathsf{Std} = \mathcal{N} \cup \bar{\mathcal{N}}$, $\mathsf{Pri} = \mathcal{U} \cup \bar{\mathcal{U}}$, $\mathsf{Vis} = \mathsf{Std} \cup \mathsf{Pri}$ and $\mathsf{Act} = \mathsf{Vis} \cup \{\tau\}$. We let $u, v, \ldots$ range over Pri, $a, b, \ldots$ 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 $\bar{S}$ denote $\{\bar{a} : a \in S\}$, where if $\bar{a} \in \bar{\mathcal{N}} \cup \bar{\mathcal{U}}$ then $\bar{\bar{a}} = a$.

Now we define processes:

Definition 1. *(cf [13, 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. $P_1 | P_2$ (parallel composition)
3. 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 $A\langle b_1, .., b_n \rangle$ comes with a defining equation $A(a_1, .., a_n) \stackrel{\text{df}}{=} P$, where P is a process whose free names are drawn from $a_1, .., a_n$. We will tend to abbreviate $a_1, .., a_n$ by $\overrightarrow{a}$. We write the empty guarded summation as 0 and abbreviate $S : \alpha.0$ by $S : \alpha$. It is assumed that the order in a summation is immaterial. We abbreviate $\emptyset : \alpha$ by α. Definition 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 $\bar{S} \cap \mathsf{Pri}$. Clearly, any names 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.

Two sublanguages of CPG are of interest:

Definition 2. *Let $\mathcal{P}_{\mathsf{Std}}$ be the sublanguage of* standard *processes generated as in Definition 1 except that all names are drawn from* Std *(i.e. we effectively take $\mathcal{U} = \emptyset$ and $S_i = \emptyset$ in clause (1)). Let $\mathcal{P}_{\mathsf{Ug}}$ be the sublanguage of* unguarded *processes generated as in Definition 1 except that all priority guards are empty (i.e. $S_i = \emptyset$ in clause (1)).*

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 names in Pri. Such processes cause no problems for strong equivalence (Proposition 4), but care is needed with weak equivalence (Sect. 7), since e.g. u and $\tau.u$ ($u \in \mathsf{Pri}$) are not weakly equivalent, as remarked above.

4 Offers and Reaction

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 3. *(cf [13, 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. $P|0 \equiv P$, $P|Q \equiv Q|P$, $P|(Q|R) \equiv (P|Q)|R$
3. new $a\ (P|Q) \equiv P|\text{new } a\ Q$ if $a \notin \mathsf{fn}(P)$;
   new $a\ 0 \equiv 0$, new a new $b\ P \equiv$ new b new $a\ P$
4. $A\langle \overrightarrow{b} \rangle \equiv \{\overrightarrow{b}/\overrightarrow{a}\}P$ if $A(\overrightarrow{a}) \stackrel{\text{df}}{=} P$

Recall that a guarded action $S:a$ is conditional on other processes in the environment not offering actions in $\bar{S} \cap \mathsf{Pri}$. 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. *By induction on* $P \in \mathcal{P}$*:*

1. $\mathsf{off}(\sum_{i\in I} S_i:\alpha_i.P_i) = \{\alpha_i : i \in I, \alpha_i \in \mathsf{Pri}, \alpha_i \notin S_i\}$
2. $\mathsf{off}(P_1|P_2) = \mathsf{off}(P_1) \cup \mathsf{off}(P_2)$
3. $\mathsf{off}(\text{new } a\ P) = \mathsf{off}(P) - \{a, \bar{a}\}$
4. $\mathsf{off}(A\langle \overrightarrow{b} \rangle) = \mathsf{off}(\{\overrightarrow{b}/\overrightarrow{a}\}P)$ if $A(\overrightarrow{a}) \stackrel{\text{df}}{=} P$

In item 1 the reason that we insist $\alpha_i \notin S_i$ is that we want to equate a process such as $u:u$ with 0, since $u:u$ can never engage in a reaction. Note that if $P \in \mathcal{P}_{\mathsf{Std}}$ then $\mathsf{off}(P) = \emptyset$.

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

Definition 5. *Let* $P \in \mathcal{P}$ *and let* $S \subseteq \mathsf{Act}$ *be finite.* P eschews S *(written* $P\ \mathsf{eschews}\ S$*) iff* $\mathsf{off}(P) \cap \bar{S} = \emptyset$.

Definition 6. *(cf [13, Definition 4.13]) The* reaction relation *on* $\mathcal{P}$ *is the smallest relation* $\to$ *on* $\mathcal{P} \times \wp(\mathsf{Pri}) \times \mathcal{P}$ *generated by the following rules:*

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

$$\frac{S:a.P + M\ \mathsf{eschews}\ T \quad T:\bar{a}.Q + N\ \mathsf{eschews}\ S}{(S:a.P + M)|(T:\bar{a}.Q + N) \to_{(S\cup T)\cap\mathsf{Pri}} P|Q} \qquad \frac{P \to_U P' \quad Q\ \mathsf{eschews}\ U}{P|Q \to_U P'|Q}$$

$$\frac{P \to_U P'}{\text{new } a\ P \to_{U-\{a,\bar{a}\}} \text{new } a\ P'} \qquad \frac{P \to_U P' \quad P \equiv Q \quad P' \equiv Q'}{Q \to_U Q'}$$

We abbreviate $P \to_\emptyset P'$ *by* $P \to P'$.

The second clause of Definition 6 is the most important. In order for an action a to react with a complementary $\bar{a}$, 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 $\mathcal{P}_{\mathsf{Ug}}$ (i.e. we let $U = \emptyset$) we recover the usual CCS reaction relation. So the new reaction relation is conservative over the old.

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 [3], which is parametrised on what set of output actions the environment is ready to perform.

Definition 7. *(cf [13, Definition 5.1]) The* transition relation *on $\mathcal{P}$ is the smallest relation $\rightarrow$ on $\mathcal{P} \times Act \times \wp(\mathsf{Pri}) \times \mathcal{P}$ generated by the following rules:*

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

$$\text{(react)} \qquad \frac{P_1 \xrightarrow{a}_{U_1} P_1' \quad P_2 \xrightarrow{\bar{a}}_{U_2} P_2' \quad P_1 \,\mathsf{eschews}\, U_2 \quad P_2 \,\mathsf{eschews}\, U_1}{P_1|P_2 \xrightarrow{\tau}_{U_1 \cup U_2} P_1'|P_2'}$$

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

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

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

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 1. *If $P \xrightarrow{\alpha}_U P'$ then $\alpha \notin U$ and U is finite. Moreover,*

$$\{u \in \mathsf{Pri} : \exists U.P \xrightarrow{u}_U\} \subseteq \mathsf{off}(P) \ .$$

To see that $\mathsf{off}(P)$ can be unequal to $\{u \in \mathsf{Pri} : \exists U.P \xrightarrow{u}_U\}$, consider $u{:}v.0|\bar{u}.0$. We see that $\mathsf{off}(u{:}v|\bar{u}) = \{v, \bar{u}\}$, but $u{:}v|\bar{u}$ cannot perform v.

As with reaction, note that if we restrict attention to the unguarded processes $\mathcal{P}_{\mathsf{Ug}}$ (i.e. we let $U = \emptyset$) we recover the usual CCS transition relation. So 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 & Winskel (Subsect. 2.1):

$$P \stackrel{\text{df}}{=} u.a + u:\bar{v} \quad Q \stackrel{\text{df}}{=} v.b + v:\bar{u} \quad R \stackrel{\text{df}}{=} \text{new } u, v\ (P|Q)$$

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

6 Strong Offer Bisimulation

Similarly to standard CCS, we 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.

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.

Definition 8. *(cf [13]) 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 $P \stackrel{\alpha}{\to}_U P'$ then for some Q' and $V \subseteq U$, we have $Q \stackrel{\alpha}{\to}_V Q'$ and $\mathcal{S}(P', Q')$

Definition 9. *Processes P and Q are* strongly offer equivalent*, written $P \stackrel{\mathsf{off}}{\sim} Q$, iff there is some strong offer bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.*

Proposition 2. *(cf [13, Prop 5.2]) $\equiv$ is a strong offer bisimulation. Hence $\equiv$ implies $\stackrel{\mathsf{off}}{\sim}$.* □

Proposition 3. *(cf [13, Theorem 5.6]) $P \stackrel{\tau}{\to}_U \equiv P'$ iff $P \to_U P'$* □

Theorem 1. *(cf [13, Proposition 5.29]) Strong offer equivalence is a congruence, i.e. if $P \stackrel{\mathsf{off}}{\sim} Q$ then*

1. $S:\alpha.P + M \stackrel{\mathsf{off}}{\sim} S:\alpha.Q + M$
2. new a $P \stackrel{\mathsf{off}}{\sim}$ new a Q
3. $P|R \stackrel{\mathsf{off}}{\sim} Q|R$
4. $R|P \stackrel{\mathsf{off}}{\sim} R|Q$

□

Note that if $P, Q \in \mathcal{P}_{\mathsf{Ug}}$ then $P \overset{\mathsf{off}}{\sim} Q$ iff $P \sim Q$, where $P \sim Q$ denotes that P and Q are strongly equivalent in the usual sense of [13]. So $\overset{\mathsf{off}}{\sim}$ is conservative over $\sim$. In fact we can say more:

Proposition 4. *Let $P, Q \in \mathcal{P}_{\mathsf{Ug}}$. If $P \sim Q$ then $C[P] \overset{\mathsf{off}}{\sim} C[Q]$, for any context $C[\cdot]$.* □

So we can reuse all the known equivalences between CCS processes when working with CPG processes.

Proposition 5. *(cf [13, Proposition 5.21]) For all $P \in \mathcal{P}$,*

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

Proposition 6. *The following laws hold:*

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

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

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

$$\begin{aligned}(\sum U_i : \alpha_i.P_i) \mid (\sum V_j : \beta_j.Q_j) \\ \overset{\mathsf{off}}{\sim} \sum\{U_i : \alpha_i.(P_i | (\sum V_j : \beta_j.Q_j)) : \forall j.\beta_j \notin \bar{U}_i\} \\ + \sum\{V_j : \beta_j.((\sum U_i : \alpha_i.P_i) | Q_j)) : \forall i.\alpha_i \notin \bar{V}_j\} \\ + \sum\{(U_i \cup V_j) : \tau.P_i | Q_j : \alpha_i = \bar{\beta}_j \in \mathsf{Vis}, \forall i', j'.\alpha_{i'} \notin \bar{V}_j, \beta_{j'} \notin \bar{U}_i\}\end{aligned} \tag{4}$$

$$\text{new } a\, (\sum U_i : \alpha_i.P_i) \overset{\mathsf{off}}{\sim} \sum((U_i - \{a, \bar{a}\}) : \alpha_i.\text{new } a\, P_i : \alpha_i \neq a, \bar{a}\} \tag{5}$$

Definition 10. *Let $\mathcal{A}_S$ be the following set of axioms: the axioms of structural congruence $\equiv$ (Definition 3) together with the five laws of Proposition 6.*

Theorem 2. *The set of axioms $\mathcal{A}_S$ is complete for $\overset{\mathsf{off}}{\sim}$ on finite CPG processes (a CPG process is* finite *if it contains no identifiers).* □

As mentioned in the Introduction, we can encode mixed input and output guarded summation using priority guards and restricted parallel composition.

Proposition 7. *Suppose that* $\{\alpha_i : i \in I\}$ *are actions which cannot react with each other, i.e. there do not exist* $i, j \in I$ *and* $a \in \mathsf{Vis}$ *such that* $\alpha_i = a$ *and* $\alpha_j = \bar{a}$*. Then*

$$\sum S_i : \alpha_i.P_i \overset{\mathsf{off}}{\sim} \text{new } u \, (\prod (S_i \cup \{u\} : \alpha_i(P_i|\bar{u})))$$

where $u \in \mathsf{Pri}$ *is some fresh name not occurring in* $\sum S_i : \alpha_i.P_i$ *and* $\prod$ *denotes parallel composition.* □

The non-reaction condition in Proposition 7 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.

7 Weak Offer Bisimulation

We now investigate weak bisimulation, where reactions are hidden.

Definition 11. $P \Rightarrow_U P'$ *iff* $P \overset{\mathrm{id}}{=} P'$ *or* $\exists U_1, \ldots, U_n . P \rightarrow_{U_1} \cdots \rightarrow_{U_n} P'$ *with* $U = U_1 \cup \cdots \cup U_n$ *(*$n \geq 1$*). We abbreviate* $P \Rightarrow_\emptyset P'$ *by* $P \Rightarrow P'$*.*

$P \overset{\alpha}{\Rightarrow}_U P'$ *iff* $\exists U', U'' . P \Rightarrow_{U'} P'' \overset{\alpha}{\rightarrow}_{U''} P''' \Rightarrow P'$ *with* $U = U' \cup U''$ *and* $\mathsf{off}(P'') \subseteq \mathsf{off}(P)$*.*

Here $P \overset{\mathrm{id}}{=} P'$ 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. The reason why we allow priority guards before performing a visible action, but not after, is as follows: For Q to simulate $P \overset{a}{\rightarrow}_U P'$, Q must expect an environment offering $\bar{U}$ 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 12. *A symmetric relation* $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ *is a* weak offer bisimulation *if* $\mathcal{S}(P, Q)$ *implies both that* $\mathsf{off}(P) = \mathsf{off}(Q)$ *and that:*

if $P \rightarrow_U P'$ *then for some* Q' *and* $U' \subseteq U$*, we have* $Q \Rightarrow_{U'} Q'$ *and* $\mathcal{S}(P', Q')$*, and for all* $a \in \mathsf{Vis}$*,*

if $P \overset{a}{\rightarrow}_U P'$ *then for some* Q' *and* $U' \subseteq U$*, we have* $Q \overset{a}{\Rightarrow}_{U'} Q'$ *and* $\mathcal{S}(P', Q')$*.*

On the sublanguage $\mathcal{P}_{\mathsf{Std}}$ (which corresponds to CCS) weak offer bisimulation is the same as for CCS [13, Proposition 6.3].

Definition 13. *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* $\mathcal{S}(P, Q)$*.*

Proposition 8. *For any P, Q, if $P \overset{\mathsf{off}}{\sim} Q$ then $P \overset{\mathsf{off}}{\approx} Q$.* □

Theorem 3. *(cf [13, Proposition 6.17]) $\overset{\mathsf{off}}{\approx}$ is a congruence.* □

So we have a congruence which conservatively extends CCS.

We now turn to the equational theory of weak offer equivalence. In CCS we have the law $P \approx \tau.P$ [13, Theorem 6.15]. However in CPG, $u \overset{\mathsf{off}}{\not\approx} \tau.u$. This is because $\mathsf{off}(u) = \{u\}$ whereas $\mathsf{off}(\tau.u) = \emptyset$. However 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 9. *(cf [13, Theorem 6.15]) The following laws hold:*

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

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

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

We stated (6), (7) and (8) because in many situations it is convenient to use conventional CCS reasoning. The next result gives the "intrinsic" τ-laws of CPG:

Proposition 10. *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 $\mathsf{off}(N + V : a.P) \subseteq \mathsf{off}(M)$:

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

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

We can derive (7) from (10) and (11). Also we can derive:

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

from (9), (10), (11). Recall that every process is strongly equivalent to a summation (Proposition 5), and so (13) is effectively as strong as (6).

Definition 14. *Let $\mathcal{A}_W$ be the axioms $\mathcal{A}_S$ (Definition 10) together with (9), (10), (11) and (12).*

Theorem 4. *The axioms $\mathcal{A}_W$ are complete for $\overset{\mathsf{off}}{\approx}$ on finite processes.* □

Proposition 11. *(cf [13, Theorem 6.19]) Unique solution of equations. Let $\overrightarrow{X}$ be a (possibly infinite) sequence of process variables X_i. Up to $\overset{\mathsf{off}}{\approx}$, there is a unique sequence $\overrightarrow{P}$ of processes which satisfy the formal equations:*

$$X_i \overset{\mathsf{off}}{\approx} \sum_j U_{ij} : a_{ij}.X_{k(ij)}$$

(notice that τs are not allowed). □

8 Example

We now revisit the interrupt example from Sect. 1. Recall that we had:

$$P \overset{\mathrm{df}}{=} \mathrm{new}\ \mathsf{mid}, \mathsf{int}_A, \mathsf{int}_B\ (A|B|I)\ A \overset{\mathrm{df}}{=} \mathsf{int}_A : a.\overline{\mathsf{mid}}.A + \mathsf{int}_A$$
$$I \overset{\mathrm{df}}{=} \mathsf{int}.(\overline{\mathsf{int}}_A.\overline{\mathsf{int}}_B + \overline{\mathsf{int}}_B.\overline{\mathsf{int}}_A) \quad B \overset{\mathrm{df}}{=} \mathsf{int}_B : b.\mathsf{mid}.B + \mathsf{int}_B$$

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

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

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), (5) to get:

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

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

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

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

Then we use law (7) to get $\tau.P + \mathsf{int} \overset{\mathsf{off}}{\approx} \tau.P$. Notice that this needs $\mathsf{off}(P) = \emptyset$, i.e. $a, b, \mathsf{int} \notin \mathsf{Pri}$. Finally:

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

By Proposition 11 we get $P \overset{\mathsf{off}}{\approx} Q$ 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.

In the Introduction we used plain equality (=) when talking about equivalence between CPG processes. This is to be interpreted as $\overset{\mathsf{off}}{\approx}$.

9 Expressiveness

In this section we show that priorities add expressive power to both CCS and the π-calculus [14,13]. As far as we are aware, this has not been previously shown for any notion of priority in process algebra. We use the work of Ene and Muntian [6], which was inspired by that of Palamidessi [16].

Definition 15. *[16] An* encoding $[\![\cdot]\!]$ *is a compositional mapping from (the processes of) one calculus to another. It is called* uniform *if* $[\![P|Q]\!] = [\![P]\!]|[\![Q]\!]$ *and, for any renaming* σ, $[\![\sigma(P)]\!] = \sigma([\![P]\!])$. *A semantics is* reasonable *if it distinguishes two processes* P *and* Q *whenever in some computation of* P *the actions on certain intended channels are different from those of any computation of* Q.

Definition 16. *(slight modification of [16, Definition 3.1]) A process*

$$P = P_1|\cdots|P_n$$

is an electoral system *if every computation of* P *can be extended (if necessary) to a computation which declares a leader* i *by outputting* $\bar{o}_i$, *and where no computation declares two different leaders.*

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, π-calculus communication is always one-one.

Theorem 5. *There is no uniform encoding of CPG into the* π*-calculus preserving a reasonable semantics.*

Proof. (Sketch) We follow the proof of Ene and Muntian's result that the broadcast π-calculus cannot be encoded in the π-calculus [6]. The network of CPG processes $P_1|\cdots|P_n$ is an electoral system, where $P_i \stackrel{\text{df}}{=} u\!:\!a.(\bar{u}|\bar{o}_i)|\bar{a}$. If P_i manages to communicate on a then P_i declares itself the leader. No other process can now do this, since P_i is preventing all the other processes from performing a by offering $\bar{u}$.

The rest of the proof is as in [6]. Suppose that we have an encoding $[\![\cdot]\!]$ of CPG into the π-calculus. Let $\sigma(o_i) = o_{m+i}$, with σ the identity otherwise. Consider $P_1|\cdots|P_{m+n}$. This is an electoral system and so the encoding $[\![P_1|\cdots|P_{m+n}]\!]$ must be also. But

$$\begin{aligned}[\![P_1|\cdots|P_{m+n}]\!] &= [\![(P_1|\cdots|P_m)|\sigma(P_1|\cdots|P_n)]\!] \\ &= [\![P_1|\cdots|P_m]\!]|[\![\sigma(P_1|\cdots|P_n)]\!]\end{aligned}$$

So we have two electoral systems of m and n processes respectively, which can be run independently in the π-calculus to produce two winners. Contradiction. □

Since CCS can be encoded in 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.

Theorem 6. *There is no uniform encoding of the π-calculus into CPG preserving a reasonable semantics.*

Proof. Much as in [16], where it is shown for CCS rather than CPG. □

The results of this section apply equally to Camilleri-Winskel and Cleaveland-Hennessy-style priority.

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 prioritised 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, we have seen that priority guards add expressiveness to both CCS and the π-calculus.

We wish to thank Philippa Gardner, Rajagopal Nagarajan, Catuscia Palamidessi, Andrew Phillips, Irek Ulidowski, Maria Grazia Vigliotti, Nobuko Yoshida and the anonymous referees for helpful discussions and suggestions.

References

1. J.C.M. Baeten, J. Bergstra, and J.-W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, 9:127–168, 1986.
2. J. Bergstra and J.-W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.
3. J. Camilleri and G. Winskel. CCS with priority choice. *Information and Computation*, 116(1):26–37, 1995.
4. R. Cleaveland and M.C.B. Hennessy. Priorities in process algebra. *Information and Computation*, 87(1/2):58–77, 1990.
5. R. Cleaveland, G. Lüttgen, and V. Natarajan. Priority in process algebra. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2001.
6. C. Ene and T. Muntian. Expressiveness of point-to-point versus broadcast communications. In *FCT '99*, volume 1684 of *Lecture Notes in Computer Science*, pages 258–268. Springer-Verlag, 1999.
7. C.J. Fidge. A formal definition of priority in CSP. *ACM Transactions on Programming Languages and Systems*, 15(4):681–705, 1993.
8. H. Hansson and F. Orava. A process calculus with incomparable priorities. In *Proceedings of the North American Process Algebra Workshop*, pages 43–64, Stony Brook, New York, 1992. Springer-Verlag Workshops in Computer Science.

9. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
10. A. Jeffrey. A typed, prioritized process algebra. Technical Report 13/93, Dept. of Computer Science, University of Sussex, 1993.
11. C.-T. Jensen. *Prioritized and Independent Actions in Distributed Computer Systems.* PhD thesis, Aarhus University, 1994.
12. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
13. R. Milner. *Communicating and Mobile Systems: the π-calculus.* Cambridge University Press, 1999.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.
15. V. Natarajan, L. Christoff, I. Christoff, and R. Cleaveland. Priorities and abstraction in process algebra. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science, 14th Conference*, volume 880 of *Lecture Notes in Computer Science*, pages 217–230. Springer-Verlag, 1994.
16. C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of the 25th Annual Symposium on Principles of Programming Languages, POPL '97*, pages 256–265. ACM, 1997.