A Calculus of Broadcasting Systems

K. V. S. Prasad*

Department of Computing Science
Chalmers University of Technology
S- 412 96 Gothenburg
Sweden
E-mail: prasad@cs.chalmers.se

September 28, 1995

Abstract. CBS is a simple and natural CCS-like calculus where processes speak one at a time and are heard instantaneously by all others. Speech is autonomous, contention between speakers being resolved non-deterministically, but hearing only happens when someone else speaks. Observationally meaningful laws differ from those of CCS. The change from handshake communication in CCS to broadcast in CBS permits several advances. (1) Priority, which attaches only to autonomous actions, is simply added to CBS in contrast to CCS, where such actions are the result of communication. (2) A CBS simulator runs a process by returning a list of values it broadcasts. This permits a powerful combination, CBS with the host language. It yields several elegant algorithms. Only processes with a unique response to each input are needed in practice, so weak bisimulation is a congruence. (3) CBS subsystems are interfaced by translators; by mapping messages to silence, these can restrict hearing and hide speech. Reversing a translator turns its scope inside out. This permits a new specification for a communication link: the environment of each user should behave like the other user.
This paper reports the stable aspects of an evolving study.

1 Introduction

Broadcast is a natural means of communication. It is the hardware primitive in local area networks [Abr70, MB76], as well as in radio and mobile telephony networks, and point-to-point message passing is implemented on top of it. But only the latter is studied in the best established theories of communication and concurrency, process calculi like CSP [Hoa85], CCS [Hen88] and ACP [BW90]. There is great interest [CNL89] in implementations of reliable broadcasting, usually on top of point-to-point communication, but little in the use of it. Most books on distributed systems, [SK88] for example, treat broadcast as a hardware feature, but not as a programming primitive. This mismatch would appear firstly to throw away a lot of communication bandwidth. This paper suggests it is also throwing away much else.

A sustained study applying process calculus techniques to broadcast communication ([Pra91, Pra93a, Pra93b, Pra94], also [Pra95, Jon93, Hol93, Pet94]) has yielded what promises to be an elegant calculus of broadcasting systems, CBS, and some elegant programs. CBS is fashioned after CCS [Mil89], but the change from handshake to broadcast communication has far reaching consequences both for programming and for theory.

This paper reports the stable aspects of CBS as of now, including some material not previously presented. It concentrates on basic theory, on programming [Pra93b] and on priority [Pra94]. The current version of CBS is a subcalculus of that of [Pra93b], and is significantly different from those of [Pra91] and [Pra93a].

Organisation of the paper. A sequential pass should encounter few forward references. Ignoring the occasional outside reference, many of Sections 3 to 11 can be read immediately after Section 2 (concepts), and any of Sections 13 to 15 after Section 12 (adding priorities to CBS). Sections 2 and 12 are basic.

For theory, the further sections are 3 (formal definition of CBS), 4 (strong bisimulation), 5 (axiomatisation), 6 (expansion theorem), 7 (weak bisimulation) and 13 (bisimulations with priorities). Familiarity with CCS is helpful here.

For exploratory programming, the further sections are 9, 15 (examples, using a simple functional language), and 10 (a CBS simulator), though even here a glance through Section 3 is recommended.

Section 8 describes a new verification paradigm, context reversal. Concepts, alternative designs, and related work are discussed in Sections 11 and 14.

2 Overview of CBS

2.1 Informal models of broadcast communication

CBS models an idealised local area network (LAN). Communication is by unbuffered broadcast, each message being instantaneously received by all nodes. Only one message can be broadcast at a time. The speaker's identity can be part of the transmitted message, but need not be. Multicasting or point to point delivery can be programmed by including in each message a header listing the intended receivers. An implementation detail invisible to LAN users, and not modelled in CBS, is the resolution of contention: if two nodes try to broadcast simultaneously, a collision occurs, no message is transmitted, and the nodes try again after a random amount of time.

Ordinary speech is another example of unbuffered broadcast communication. Everyday conversation does not enforce speaking in turns or offer anonymity since speakers can be recognised by their voices, but a public address system (PA), such as in an airport, does. Anyone can hand in messages to be read out, one at a time, in a sequence chosen by the announcer. Flight announcements usually mention neither source nor audience, both being implicitly clear. "Would A please meet B at the desk" indicates both source and intended listener. After this message, A would probably remove any pending paging request for B.

CBS formalises the idealised LAN or PA using concepts from process calculus.

2.2 Formal modelling concepts from process calculus

The behaviour of a process (or system) consists of communication actions: the transmission and reception of messages. The environment, or observer, of a system p is itself a process communicating with p, i.e., in parallel with it. The transitions

$$p \xrightarrow{w!} p' \qquad\qquad p \xrightarrow{w?} p'$$

say that p can transmit (resp. receive) the message w and become process p' in doing so. The process $p \,|\, q$ consists of the processes p and q composed in parallel. The meaning of operators, such as "$|$", is given by inference rules such as

$$\frac{p \xrightarrow{w!} p' \quad q \xrightarrow{w?} q'}{p \mid q \xrightarrow{w!} p' \mid q'}$$

each of which says that if its premises (here $p \xrightarrow{w!} p'$ and $q \xrightarrow{w?} q'$) hold, then so does its conclusion (here $p \mid q \xrightarrow{w!} p' \mid q'$). Here p transmits w and q receives it, and they evolve together to p' and q'. An observer of $p \mid q$ sees it transmit w.

Processes are regarded as equal if they have the same behaviour. Following [Mil89], this paper defines p and q to have the same behaviour (notation $p \sim q$) if there exists a *bisimulation relation* $\mathcal{R}$ over processes such that $p\mathcal{R}q$. See Section 4 for definitions. A less discriminating equivalence, written $\approx$, is defined via *weak bisimulation* in Section 7. It is expected that $p \mid q \sim q \mid p$, and that $(p \mid q) \mid r \sim p \mid (q \mid r)$. That is, $\mid$ is expected to be commutative and associative.

To build systems out of subsystems, all calculi provide some static scoping that allows the internal actions of a subsystem p to be hidden from its environment e, and restricting the set of actions via which p interacts with e.

2.3 CBS without scoping

The design decisions and the operators of CBS are presented here and in the next subsection. CBS models the nodes in a network (or users of a PA) as processes in parallel. Note that the announcer of the PA corresponds to the collision resolution mechanism in a LAN, and is not a process.

Any CBS process says and hears messages of just one type, which does not change as the process evolves. Processes can only be put in parallel with others that speak the same type. Now $\xrightarrow{5!}$ and $\xrightarrow{5?}$ stand for the actions of saying and hearing "5". Note the absence of channel names or other addresses. These can be included in the message if needed; for example, by letting messages be pairs such as $\langle a, 5 \rangle$ where a is a channel name.

Parallel composition. Let w be any message. The inference rules of $\mid$ are

$$\frac{p \xrightarrow{w!} p' \quad q \xrightarrow{w?} q'}{p \mid q \xrightarrow{w!} p' \mid q'} \qquad \frac{p \xrightarrow{w?} p' \quad q \xrightarrow{w!} q'}{p \mid q \xrightarrow{w!} p' \mid q'} \qquad \frac{p \xrightarrow{w?} p' \quad p \xrightarrow{w?} q'}{p \mid q \xrightarrow{w?} p' \mid q'}$$

3

Transmissions cannot be combined. Nor is there interleaving: from $p \xrightarrow{w!} p'$ it does *not* follow that $p \mid q \xrightarrow{w!} p' \mid q$ for any q. Clearly, $\mid$ is commutative. The derivations below illustrate associativity and one-to-many communication.

$$
\dfrac{\dfrac{p \xrightarrow{w!} p' \quad q \xrightarrow{w?} q'}{p \mid q \xrightarrow{w!} p' \mid q'} \quad r \xrightarrow{w?} r'}{(p \mid q) \mid r \xrightarrow{w!} (p' \mid q') \mid r'}
\qquad
\dfrac{p \xrightarrow{w!} p' \quad \dfrac{q \xrightarrow{w?} q' \quad r \xrightarrow{w?} r'}{q \mid r \xrightarrow{w?} q' \mid r'}}{p \mid (q \mid r) \xrightarrow{w!} p' \mid (q' \mid r')}
$$

Both p and q have to act, on the same w, for $p \mid q$ to act. So $w?$ can be read as allowing the environment to say "w". But the environment can say anything at any time. So CBS ensures that every process is *input enabled* (input/output automata [LT87, Seg92] have a similar property): every process q has a $w?$ action for every w, even if only a self loop $q \xrightarrow{w?} q$. This yields apparent interleaving.

$$
\dfrac{p \xrightarrow{w!} p' \quad q \xrightarrow{w?} q}{p \mid q \xrightarrow{w!} p' \mid q}
$$

More generally, input enabling ensures that if p has a speech action, so does $p \mid q$.

Nil and Prefixes. The process **0** (pronounced "nil") says nothing, and ignores everything it hears.

$$
\mathbf{0} \xrightarrow{w?} \mathbf{0}
$$

Thus $p \mid \mathbf{0}$ behaves like p. The process $w! p$ wishes to say w and become p, but must also be prepared to hear any message w'. It too ignores what it hears.

$$
w! p \xrightarrow{w!} p \qquad\qquad w! p \xrightarrow{w'?} w! p
$$

The process $x? (x + 1) ! q$ listens. If it hears v, it evolves to $(x + 1) ! q$ with v substituted for x. The "+" shows that v must be a number. Note that subsequent utterances may reveal that a listener heard something, and even what it heard.

$$
x? (x + 1) ! q \xrightarrow{v?} ((x + 1) ! q) [v/x] \equiv (v + 1) ! q
$$

Conditionals and Constant Definitions. Both are illustrated by the derived construct $v? p$. Its definition and behaviour are

$$
v? p \stackrel{\mathrm{def}}{=} x? \text{ if } x = v \text{ then } p \text{ else } v? p
$$

$$
v? p \xrightarrow{v?} p \qquad\qquad v? p \xrightarrow{w?} v? p \qquad \text{if } w \neq v
$$

Responding differently to different inputs is done by nested conditionals.

Choice. Some processes wish to speak but also listen. The operator "&" takes a specification of response to input, and one offer of output. The process $x? p \,\&\, w! q$ will say w and become q, except if preempted by the environment.

$$
x? p \,\&\, w! q \xrightarrow{w!} q \qquad\qquad x? p \,\&\, w! q \xrightarrow{v?} p[v/x]
$$

"&" is the only form of choice in CBS! Thus processes have a unique response to each input. They may however have more than one output.

$$
2! p \mid 7! q \xrightarrow{2!} p \mid 7! q \qquad\qquad 2! p \mid 7! q \xrightarrow{7!} 2! p \mid q
$$

Speakers in parallel are the only source of non-determinism in CBS.

2.4 Static scoping

A translator ϕ is specified by a pair of total functions $\langle \phi^\uparrow, \phi_\downarrow \rangle$, which map messages from, say, booleans to integers and vice-versa. With these types, ϕ is a function that maps a boolean speaking process p to an integer speaking one ϕp. If p says u, then ϕp says $\phi^\uparrow u$, and if ϕp hears w, then p hears $\phi_\downarrow w$.

Example 1. Below, p is polymorphic, and q is an integer speaking process.

$$
\begin{aligned}
p &= x?\, x!\, \mathbf{0} \\
q &= \phi\ p \\
\phi^\uparrow\, x &= \text{if } x \text{ then } 1 \text{ else } 0 \\
\phi_\downarrow\, x &= \textit{odd } n
\end{aligned}
$$

p is instantiated in q to a boolean speaker.

Hiding and restriction. Let τ be a special silent message that can be spoken by any process, no matter what type of message it speaks. τ is ignored by all processes:

$$p \xrightarrow{\tau?} p$$

The domains and ranges of translating functions are also augmented by τ, with the constraint $\phi^\uparrow \tau = \phi_\downarrow \tau = \tau$. Suppose $\phi^\uparrow 5 = \tau$ and $p \xrightarrow{5!} p'$. Then $\phi p \xrightarrow{\tau!} \phi p'$, so that the 5! is *hidden*: it does not disturb the environment. Similarly, if $\phi_\downarrow 6 = \tau$ then $\phi p \xrightarrow{6?} \phi p$, so that the 6 is *restricted*: the environment can say it without disturbing p.

Example 2. Suppose u_0 and u_1 are α speakers wishing to use a connecting medium M that carries pairs $\langle w, x \rangle$ where $w \in \{0, 1\}$ and $x : \alpha$. Suppose M echoes anything it hears, but with the tag changed from w to $1 - w$.

The users could be modified to tag their messages accordingly. A more structured solution is to leave them as they are, to define translators

$$\phi^\uparrow_w\, x = \langle w, x \rangle \qquad\qquad \phi_{w\downarrow} \langle w', x \rangle = \text{if } w = w' \text{ then } x \text{ else } \tau$$

where $w, w' \in \{0, 1\}$, and to make the system $\phi_0 u_0 \mid M \mid \phi_1 u_1$. Then ϕ_w tags the speech of u_w with w, and lets it hear only speech with that tag.

Reversal. The process $e \mid \phi p$ is described from the location of e, in the sense that e is observed untranslated, but p is observed through ϕ. If p says u, the observer hears $\phi^\uparrow u$, and if e or the observer says w, then p hears $\phi_\downarrow w$. Is there a description from p's location? The new observer sees p untranslated. If e says w, the new observer hears $\phi_\downarrow w$, and if the observer or p says u, then e hears $\phi^\uparrow u$. That is, e is observed through ϕ^R where ϕ^R is specified by $\langle \phi_\downarrow, \phi^\uparrow \rangle$. The new description is therefore $\phi^R e \mid p$. The reversibility of translators is the key to describing a system from various viewpoints. Such descriptions are useful in formulating correctness requirements.

2.5 Audibility and autonomy

Is $\xrightarrow{5!}$ a good representation of speech and $\xrightarrow{5?}$ one of hearing? To answer this, some terminology is needed. If a process gives out information in doing an action, the action is *audible*; otherwise it is *silent*. If a process needs information from its environment to do an action, the action is *controlled*; otherwise it is *autonomous*.

In the informal models, hearing is silent. It is also controlled, for the action of hearing "5" is delayed until the environment says "5" (the information needed is synchronisation). Speech is instantaneously audible to everyone. It is also autonomous, for the message is already available to the intending speaker (which in speaking does learn that its message was chosen, but can conclude nothing about the environment).

It is now possible to check that the rules of CBS are consistent with these requirements. For example, consider the rule

$$\frac{p \xrightarrow{w!} p' \quad q \xrightarrow{w?} q'}{p \mid q \xrightarrow{w!} p' \mid q'}$$

That q contributed a $w?$ is no news because of input enabling. So hearing is silent, p learns nothing, and speech is autonomous. Hearing is also controlled, and speech audible, for q learned what was said. (There is also a q'' such that $q \xrightarrow{w'?} q''$ but this can only contribute to an action of $p \mid q$ if p has a $w'!$ action.)

The prefixes conform too: the w in $w! p \xrightarrow{w!} p$ is known in advance, while the v in $x? (x+1)! q \xrightarrow{v?} (v+1)! q$ is learnt in the action.

Finally, a translator can reduce the amount of information flowing in or out, but cannot reverse the flow. Thus the translation rules conform. In the extreme cases, speech is hidden, and hearing restricted.

2.6 Programming

Running a process. One way to run a process p is to interact with it from a keyboard. But an interactive simulator [Kor94] is a tool for illustration, not for running programs with thousands of actions. The interaction can be done in "batch" mode by preprogramming the user's responses as e, but now $e \mid p$ is itself a process to be run, so the question remains.

Now consider $p \stackrel{\text{def}}{=} x? \mathbf{0} \& 5! \mathbf{0}$. It will say 5 or hear v if its environment says v. But if the environment is silent then p *will* say 5. Similarly, $p \stackrel{\text{def}}{=} (x? \mathbf{0} \& 5! \mathbf{0}) \mid 6! \mathbf{0}$ can in general do the sequence of actions $\xrightarrow{3?}\xrightarrow{6!}$, but with a silent environment will only do $\xrightarrow{5!}\xrightarrow{6!}$ or $\xrightarrow{6!}$.

Definition 1. A *run* of p is a maximal sequence of autonomous actions that p can perform. A run $\xrightarrow{w_1!}\cdots\xrightarrow{w_n!}\cdots$ can be represented by the list $[w_1, \ldots, w_n, \ldots]$.

Removing (only) the τ's from a run yields an *audible run*. An audible run of a process is a maximal sequence that a silent environment can hear.

6

A Simulator for CBS. The simulator provides a function *run* that takes a process p and returns an audible run of p. This enables an interesting and powerful programming paradigm. This paper gives several examples, including some new algorithms. Experimentation with programming has strongly influenced the development of CBS. All the examples have been run on the simulator, implemented in SML and Haskell. The simulator is fairly efficient and very useable; it is also very simple.

Host languages. As is standard for value-passing process calculi, CBS is formally presented with constant definitions, conditionals, and substitution of data values for variables. This last is seen in $x? \, p \xrightarrow{v?} p[v/x]$. The examples use instead the rule $?f \xrightarrow{v?} f \, v$, where f is any ML or Haskell function from data to processes. Conditionals are then just a special kind of function. The simulator even borrows constant definitions from the host language. The resulting user language is a combination of CBS with ML or Haskell, well-typed, natural and powerful. Further, since a run is just a list of data values, and processes can be parameterised, it is possible to mix concurrent and sequential programming. If the host language has parallel evaluation, CBS can be seen as a high level annotation language for such evaluation.

Formally linking the host language with CBS is a topic of ongoing work. Nonetheless, the examples are reported because the concepts seem simple and are now quite old.

2.7 Priority

Priorities are a well known programming tool in concurrent systems, and have obvious meaning in broadcast systems: the airport announcer can give flight announcements, say, priority over personal messages. It makes sense to attach priorities to autonomous actions, but little sense to attach priority to controlled actions—a process might prefer to hear 5 rather than 6, but this is irrelevant, since the environment says what it likes.

The following ideas are enough to extend CBS with priorities; the result is called PCBS. The priority of a process is defined to be that of its highest priority transmission. Processes hear speech at priority greater than or equal to their own, and refuse the rest. Since speech must be heard by everyone, this refusal ensures that the lower priority speech will not happen. The changes to CBS are minimal, yet the language gains significantly in power.

Termination detection. A CBS process cannot detect when its environment has fallen silent. But termination of high priority activity can be detected by succeeding in speaking at low priority. This is a recurring theme in the use of PCBS, and leads to novel uses of priority far beyond simple interrupts.

PCBS is strictly more powerful than CBS. With only the unit data type, CBS is essentially useless: there is no obvious way to get information in and out of systems. Unary coding is not possible because the ends of code sequences cannot be detected. But this is possible in PCBS, by saying the code sequence at high priority, while the receiver attempts to speak at low priority.

Let α be a datatype. Let τ be a distinguished value, $\tau \notin \alpha$, and α_τ be $\alpha \cup \{\tau\}$. Let $x : \alpha$ be a variable and $w : \alpha_\tau$ an expression.
Let β be another datatype. Let ϕ be specified by $\phi^\uparrow : \beta_\tau \to \alpha_\tau$ and $\phi_\downarrow : \alpha_\tau \to \beta_\tau$ satisfying $\phi^\uparrow \tau = \tau$ and $\phi_\downarrow \tau = \tau$.
Let b be a boolean expression. Then the elements of $Proc\ \alpha$ are given by

$$p ::= \mathbf{0} \ \Big| \ !s \ \Big| \ ?f \ \Big| \ f \ \& \ s \ \Big| \ p|p \ \Big| \ \phi p_\beta \ \Big| \ \text{if } b \text{ then } p \text{ else } p \ \Big| \ A\ d$$

where $f ::= [x]p$, $s ::= \langle w, p \rangle$, $p_\beta : Proc\ \beta$, and A ranges over constants, declared in (mutually) recursive guarded definitions $A\ z \overset{\text{def}}{=} p$, parameterised by a datatype ranged over by variable z and expression d.
The semantics below also uses p_i, p', $p'_i : Proc\ \alpha$, $p'_\beta : Proc\ \beta$, and values $v : \alpha$ and $u : \beta_\tau$.

Tau	$p \xrightarrow{\tau ?} p$		
Guarded	$!\langle w, p \rangle \xrightarrow{w!} p$		$f \ \& \ \langle w, p \rangle \xrightarrow{w!} p$
Sum	$\mathbf{0} \xrightarrow{v?} \mathbf{0} \qquad !s \xrightarrow{v?} !s \qquad ?[x]p \xrightarrow{v?} p[v/x]$		$[x]p \ \& \ s \xrightarrow{v?} p[v/x]$

Compose[1]	$\dfrac{p_1 \xrightarrow{w \natural_1} p'_1 \quad p_2 \xrightarrow{w \natural_2} p'_2}{p_1|p_2 \xrightarrow{w(\natural_1 \bullet \natural_2)} p'_1|p'_2} \ \natural_1 \bullet \natural_2 \neq -$	$\begin{array}{c|cc} \bullet & ! & ? \\ \hline ! & - & ! \\ ? & ! & ? \end{array}$

Translate	$\dfrac{p_\beta \xrightarrow{u!} p'_\beta}{\phi p_\beta \xrightarrow{\phi^\uparrow u!} \phi p'_\beta}$	$\dfrac{p_\beta \xrightarrow{\phi_\downarrow w?} p'_\beta}{\phi p_\beta \xrightarrow{w?} \phi p'_\beta}$

Conditional[1]	$\dfrac{p_1 \xrightarrow{w\natural} p'_1}{\text{if true then } p_1 \text{ else } p_2 \xrightarrow{w\natural} p'_1}$	$\dfrac{p_2 \xrightarrow{w\natural} p'_2}{\text{if false then } p_1 \text{ else } p_2 \xrightarrow{w\natural} p'_2}$

Define[1] $A\ z \overset{\text{def}}{=} p$	$\dfrac{p[d/z] \xrightarrow{w\natural} p'}{A\ d \xrightarrow{w\natural} p'}$

[1] $\natural$, $\natural_1$, $\natural_2$ range over $\{!, ?\}$. $-$ means "undefined" in the synchronisation algebra for $\bullet$.

Table 1. The syntax of CBS and the semantics of closed processes

3 The Syntax and Semantics of CBS

The syntax and communication actions of CBS processes are given in Table 1. CBS is a framework or a coordination language, not a complete programming language. No syntax or computation rules are given for data expressions. The user chooses these for the application at hand. The evaluation of data is not represented, but is assumed to terminate, and closed data expressions merely stand for their values. Non-terminating evaluations are discussed briefly in Section 10.

Abbreviation	Meaning
$x?\,p$	$?[x]p$
$w!\,p$	$!\langle w,p\rangle$
$x?\,p\ \&\ s$	$[x]p\ \&\ s$
$f\ \&\ w!\,p$	$f\ \&\ \langle w,p\rangle$
$x?\,p\ \&\ w!\,p$	$[x]p\ \&\ \langle w,p\rangle$
$v?\,p$	X, where $X = x?$ if $x = v$ then p else X
$\prod_{i\in I}p_i$	$p_1\,|\ldots|\,p_n$, where 1 to n are the elements of I.
$A \stackrel{\text{def}}{=} p$	$A\ () \stackrel{\text{def}}{=} p$ where $()$ is the only element of the type "Unit".

This paper often uses $f\ v$ for the function application $f\,(v)$.

NOTE the abuse of notation in the third, fourth and fifth lines. In the context of the operator $\&$ the subexpressions $x?\,p$ and $w!\,p$ stand for the abstraction $[x]p$ and the pair $\langle w,p\rangle$ respectively; outside of a $\&$ context, they stand for the processes $?[x]p$ and $!\langle w,p\rangle$ respectively.

A translator ϕ can be specified by the graphs of $\phi^{\uparrow}$ and $\phi_{\downarrow}$. This is done in the form $\{\ldots, u_i \uparrow w_i, \ldots, w_j \downarrow u_j, \ldots\}$, where the elements $u \uparrow w$ specify $\phi^{\uparrow}$; the elements $w \downarrow u$ specify $\phi_{\downarrow}$. Domain elements not mentioned are mapped to τ.

Table 2. Syntactic abbreviations

Types. Given the datatype α, the syntax defines inductively the set *Proc* α. Not every type α is permissible here: it must be possible to determine when two elements of α are equal. Further, α may not itself involve the type *Proc* β for any β, that is, this paper is restricted to first order CBS.

A translator $\phi: Proc\ \beta \to Proc\ \alpha$ is specified by a pair of functions $\phi^{\uparrow}: \beta_{\tau} \to \alpha_{\tau}$ and $\phi_{\downarrow}: \alpha_{\tau} \to \beta_{\tau}$. That is, $\phi p \equiv \mathsf{Trans}\ \langle \phi^{\uparrow}, \phi_{\downarrow}\rangle\ p$, where Trans is the translation operator, always dropped in writing. Informally, $\_^{\uparrow}$ and $\__{\downarrow}$ are treated as projections yielding the components $\phi^{\uparrow}$ and $\phi_{\downarrow}$ of a translator ϕ. Different translations are possible between the same pairs of types. If $p': Proc\ \alpha$ and $p' \equiv \phi p$ for some ϕ and some p, then $\exists \beta$ such that $p: Proc\ \beta$ and $\phi: Proc\ \beta \to Proc\ \alpha$, but what β is remains unknown. This is an example of an *existential type* [MP85].

Abbreviations. The reader is invited to check that the syntactic abbreviations in Table 2 are consistent with the operational definitions in Section 2.

Definition 2 Reverse of a translator. If ϕ is a translator specified by $\langle \phi^{\uparrow}, \phi_{\downarrow}\rangle$, then the *reverse* of ϕ, denoted ϕ^R, is the translator specified by $\langle \phi_{\downarrow}, \phi^{\uparrow}\rangle$.

Defining equations and guardedness. This paper assumes that there is an unmentioned set of guarded constant definitions available for use in process terms.

Definition 3 Guardedness. Processes of the form $\mathbf{0}$, $!s$, $?f$ or $f\&s$ are guarded. If p and q are guarded, so are $p\,|\,q$ and if b then p else q. If p is guarded, so is ϕp. If $A\ z \stackrel{\text{def}}{=} p$ then $A\ d$ is guarded if $p[d/z]$ is. A definition $A\ z \stackrel{\text{def}}{=} p$ is guarded if $p[d/z]$ is guarded for all d.

9

Examples of unguarded definitions are $p \stackrel{\text{def}}{=} p$, $p \stackrel{\text{def}}{=} \phi p$ and $p \stackrel{\text{def}}{=} 2! \, q \mid p$. Since every definition in this paper is guarded, so is every process.

Induction on the structure of processes does not work for constant applications, because to prove the hypothesis for $A \, d$, the starting point is $p[d/z]$, which is not smaller than $A \, d$. But the proof of guardedness of $p[d/z]$ is smaller than that of $A \, d$. Induction can therefore be carried out on the depth of proof of guardedness of processes (abbreviated "induction on guardedness").

Open and closed processes. Let $x{:}\alpha$ be a (data) variable. Occurrences of x in p become *bound* in the *process abstraction* $[x]p$, and the scope of x in $[x]p$ is p. Bound variables are assumed to be renamed as necessary to avoid clashes under substitution. A process is *closed* if it has no free variables, and is *open* if it does. Thus $x? \, x! \, \mathbf{0}$ is closed while $x! \, \mathbf{0}$ is open. The set of all (open) processes is denoted $\mathbf{P}$, and the set of closed processes $\mathbf{P}_{cl}$.

Let $v{:}\alpha$ be a (data) value and let $p[v/x]$ denote the result of substituting v for x in p. The user has to supply the functions that substitute values for variables in data expressions. These functions extend from α to $\mathbf{P}$ in the evident way. For example, $(w! \, p) \, [v/x] \;=\; (w[v/x]) \, ! \, p[v/x]$ and $(\text{if } b \text{ then } p_1 \text{ else } p_2) \, [v/x] \;=\; \text{if } b[v/x] \text{ then } p_1[v/x] \text{ else } p_2[v/x]$.

Only closed processes can communicate. Open ones cannot, by definition.

Communication actions. For each $w{:}\alpha_\tau$, there are relations $\xrightarrow{w!}$ and $\xrightarrow{w?}$ over *Proc* α. These are the least relations satisfying the axioms and inference rules in Table 1. It is convenient to let $\natural$, $\natural_1$ and $\natural_2$ be variables ranging over $\{! \, , ? \, \}$.

Guarded sums. CBS has no general choice operator, only a *guarded sum* of the form $f \, \& \, s$. This is *not* a CCS-style sum "$?f + !s$" of $?f$ and $!s$, which would have the derivation below. The guarded sum $f \, \& \, s$ has no such behaviour.

$$\frac{!s \xrightarrow{v?} !s}{?f + !s \xrightarrow{v?} !s}$$

Section 6 presents an extended guarded sum $f \, \& \, \{s_i \mid i \in I\}$, needed (only) for an expansion theorem. Of this extended sum, $?f$ and $f \, \& \, s$ are the cases where I is empty or a singleton. Section 4 shows that $\mathbf{0}$ and $!s$ can be derived by recursion from $?f$ and $f \, \& \, s$ respectively. So $\mathbf{0}$, $!s$, $?f$ and $f \, \& \, s$ can all be seen as special cases of the extended sum.

3.1   Properties of the calculus

Let $p \xrightarrow{w\natural}$ mean "$\exists \, p'$ such that $p \xrightarrow{w\natural} p'$", and let $p \xrightarrow{w\natural}\!\!\!\!\!/\ $ mean "$\not\exists \, p'$ such that $p \xrightarrow{w\natural} p'$". The propositions below confirm that CBS is well behaved.

Proposition 4 Input enabling and determinism. $\forall p, \, w, \, \exists ! \, p'$ *such that* $p \xrightarrow{w?} p'$.

Proof. By induction on the guardedness of p. For ϕp, note that $\phi_\downarrow$ is total.

Definition 5. p/w, the *image* of p under w, is the unique p' such that $p \xrightarrow{w?} p'$.

p/w can be recursively computed by

p/τ	$= p$	$(p \mid q)/v$	$= (p/v) \mid (q/v)$
$\mathbf{0}/v$	$= \mathbf{0}$	$(\phi p)/v$	$= \phi\,(p/\phi_\downarrow v)$
$!s/v$	$= !s$	(if b then p else q)$/v$	$=$ if b then p/v else q/v
$?f/v = (f \,\&\, s)/v = f\, v$		$(A\, d)/v$	$= (p[d/z])/v$ if $A\, z \stackrel{\text{def}}{=} p$

Proposition 6 Finite output branching. $\forall p$, *the set* $\{w \mid p\xrightarrow{w!}\}$ *is finite.*

Proof. By induction on the guardedness of p.

Proposition 7 Image finite. $\forall p$, w, $\natural$, *the set* $\{p' \mid p \xrightarrow{w\natural} p'\}$ *is finite.*

Proof. Given p and $w?$, there is only one p'. Given p and $w!$, use induction on guardedness. $\mathbf{0}$, $?f$, $!s$ and $f \,\&\, s$ have at most one p'. By hypothesis there are only finitely many p_1' such that $p_1 \xrightarrow{w!} p_1'$ yielding $p_1 \mid p_2 \xrightarrow{w!} p_1' \mid (p_2/w)$. For $\phi p \xrightarrow{w!} \phi p'$, there are only finitely many w' such that $\phi^\uparrow w' = w$ and $p \xrightarrow{w'!} p'$, by the previous proposition. If $A\, z \stackrel{\text{def}}{=} p$, then $A\, d \xrightarrow{w\natural} p'$ iff $p[d/z] \xrightarrow{w\natural} p'$, and there are only finitely many p' by hypothesis.

To derive $5!\,\mathbf{0} \mid x?\,3!\,\mathbf{0} \xrightarrow{5!} \mathbf{0} \mid 3!\,\mathbf{0}$, the premise $x?\,3!\,\mathbf{0} \xrightarrow{5?} 3!\,\mathbf{0}$ is needed. But to derive any $\xrightarrow{5?}$ transition, no premises are needed that involve a $\xrightarrow{v!}$ for any v. The next proposition formulates this; for a discussion, see Section 11.

Proposition 8. *For all w and w', $\xrightarrow{w?}$ transitions can be derived independently of $\xrightarrow{w'!}$ transitions.*

Proof. The rules for $\mathbf{0}$ and guarded sum have no premises. The translate, conditional and recursion rules yield a ? action in the conclusion with only a ? action in the premise. For $\mid$, a $w?$ results only if both components do a $w?$ action.

4 Strong Bisimulation

Definition 9 Strong bisimulation for closed processes. $\mathcal{R} \subseteq \mathbf{P}_{cl} \times \mathbf{P}_{cl}$ is a strong bisimulation if whenever $p\mathcal{R}q$,
(i) if $p \xrightarrow{w\natural} p'$ then $\exists q'$ such that $q \xrightarrow{w\natural} q'$ and $p'\mathcal{R}q'$,
(ii) if $q \xrightarrow{w\natural} q'$ then $\exists p'$ such that $p \xrightarrow{w\natural} p'$ and $p'\mathcal{R}q'$

The largest strong bisimulation is an equivalence, denoted $\sim$. To show $p \sim q$, find a bisimulation $\mathcal{R}$ such that $p\mathcal{R}q$. All the laws in the proposition below are shown this way. It is sometimes easier to find a relation $\mathcal{R}$ that is a *strong bisimulation upto* $\sim$, i.e., if it satisfies the above definition with $p' \sim \mathcal{R} \sim q'$ instead of $p'\mathcal{R}q'$ at the end of (i) and (ii). Then too $p\mathcal{R}q$ implies $p \sim q$.

This paper does not use Hennessy-Milner logic, but note that the modal characterisation of bisimulation [Mil89] goes through if $\langle w\natural \rangle$ is defined by

$$p \models \langle w\natural \rangle\, \mathsf{A} \text{ iff for some } p',\ p \xrightarrow{w\natural} p' \text{ and } p' \models \mathsf{A}$$

11

Proposition 10 Strong bisimulation laws.

1. (a) $x?\,\mathbf{0} \sim \mathbf{0}$

   (b) $(x?\,w!\,p)\,\&\,(w!\,p) \sim w!\,p$

2. (a) $\mathbf{0} \sim X$ where $X \stackrel{\text{def}}{=} x?X$

   (b) $w!\,p \sim X$ where $X \stackrel{\text{def}}{=} x?X\,\&\,w!\,p$

3. $(\mathbf{P}/\sim, |, \mathbf{0})$ is a commutative monoid.

4. (a) $\phi\mathbf{0} \sim \mathbf{0}$

   (b) $\phi\,(w!\,p) \sim \left(\phi^{\uparrow}w\right)!\,\phi p$

   (c) $\phi\,(x?\,p) \sim y?\,\phi\,(\text{if } \phi_{\downarrow}y = \tau \text{ then } x?\,p \text{ else } p[\phi_{\downarrow}y/x])$

   (d) $\phi\,(x?\,p\,\&\,w!\,q) \sim y?\,\phi\,(\text{if } \phi_{\downarrow}y = \tau \text{ then } x?\,p\,\&\,w!\,q \text{ else } p[\phi_{\downarrow}y/x])$
   $$\&\,\left(\phi^{\uparrow}w\right)!\,\phi q$$

5. Let $f\ X = [y]\,(\text{if } \phi_{\downarrow}y = \tau \text{ then } X \text{ else } \phi\,(p[\phi_{\downarrow}y/x]))$

   (a) $\phi\,(x?\,p) \sim X$ where $X \stackrel{\text{def}}{=} ?\,(f\ X)$

   (b) $\phi\,(x?\,p\,\&\,w!\,q) \sim X$ where $X \stackrel{\text{def}}{=} (f\ X)\,\&\,\left(\phi^{\uparrow}w!\,\phi q\right)$

6. $\phi\,(\psi p) \sim (\phi \circ \psi)\,p$ where $\phi \circ \psi$ is specified by $\langle \phi^{\uparrow} \circ \psi^{\uparrow}, \psi_{\downarrow} \circ \phi_{\downarrow}\rangle$.

7. $\phi\,(p_1 \mid p_2) \sim \phi p_1 \mid \phi p_2$ if $\forall v \in \mathbf{L}_1 \cup \mathbf{L}_2,\ \phi_{\downarrow}\phi^{\uparrow}v = v$, where p_i has sort $\mathbf{L}_i$.

Definition 11. Let $p\colon Proc\ \alpha$. For any $\mathbf{L} \subseteq \alpha$, if the transmissions $w!$ of p and all its derivatives are such that $w \in \mathbf{L} \cup \{\tau\}$ then p has sort $\mathbf{L}$, written $p\colon \mathbf{L}$.

The sort describes only output since p can always input any element of α. To describe sorts as types would need a form of subtyping, not available in ML.

Law 2 shows that $\mathbf{0}$ and $w!\,p$ encapsulate a particular recursion. They could be regarded as derived operators, but are retained as primitives to provide some non-recursive processes, and a base for proofs by induction. Similarly, laws 5(a) and (b) show that $\phi\,(x?\,p)$ and $\phi\,(x?\,p\,\&\,w!\,q)$ encapsulate recursion.

CBS has no laws corresponding to the CCS + laws of associativity, commutativity, idempotence, and $\mathbf{0}$-identity, because the guarded sums have at most one output branch, and their input behaviour is deterministic.

$\sim$ is extended to process abstractions: $[x]p \sim [x]q$ iff $\forall v.\,p[v/x] \sim q[v/x]$. From $[x]p \sim [x]q$, it follows that $x?\,p \sim x?\,q$ and that $x?\,p\,\&\,s \sim x?\,q\,\&\,s$.

Example 3. It is easy to show that $q \sim r$ below.

$$q = \phi\,(x?\,x!\,\mathbf{0})$$
$$\phi = \{\text{true} \uparrow 1, \text{false} \uparrow 0, n \downarrow odd\ n\}$$
$$r = n?\,(n\ mod\ 2)\,!\,\mathbf{0}$$

Proposition 12. $\sim$ *is a congruence for CBS*

Proof. By adapting the corresponding proof in [Mil89]. Only one case merits mention. To show that $p \sim q$ implies $\phi p \sim \phi q$, show that $\mathcal{R} = \{\langle \phi p, \phi q\rangle \mid p \sim q\}$ is a bisimulation. If $\phi p \xrightarrow{w'!} p''$ then $p'' \equiv \phi p'$ for some p', and $\exists w$ such that $\phi^{\uparrow}w = w'$ and $p \xrightarrow{w!} p'$. For each such w, since $p \sim q$, $\exists q'$ such that $q \xrightarrow{w!} q'$ and $p' \sim q'$. Then $\phi q \xrightarrow{\phi^{\uparrow}w!} \phi q'$, and $\langle \phi p', \phi q'\rangle \in \mathcal{R}$. The case $\phi p \xrightarrow{w?} p''$ is similar.

The processes generated by $\mathbf{0}$, !, ?, & and conditionals are called "finite guarded sums". For these, Laws 1(a) and (b) are the only axioms in a complete axiomatisation of strong bisimulation (Section 5).

5 Proof system for finite guarded sums

The proof systems [Mil89] for a pure calculus consist of a set of axioms for the equality at hand, and inference rules for reflexivity, symmetry and transitivity of this relation and to allow the substitution of equals for equals in various contexts. These inference rules are so obvious that sometimes no mention is made of them.

For a value passing calculus, a proof system also needs inference rules (see Table 3) to permit reasoning about process abstractions (such as the p in $x? p$), to allow reasoning about data to be integrated with reasoning about processes, and to resolve conditional process terms. See [HL93] for rather different inference rules, to deal with value passing CCS.

Let x, $v\!:\!\alpha$, $w\!:\!\alpha_\tau$ and $f::=[x]p$ and $s::=\langle w,p\rangle$. In this section, processes p, p_i, p' etc. are restricted to *finite guarded sums*, whose syntax is given by

$$p ::= \mathbf{0} \ \Big| \ !s \ \Big| \ ?f \ \Big| \ f \ \& \ s \ \Big| \ \text{if } b \text{ then } p \text{ else } p$$

Congruence	$\dfrac{\vdash f_1 = f_2 \quad \vdash w_1 = w_2 \quad \vdash p_1 = p_2}{\vdash f_1 \ \& \ w_1! \, p_1 = f_2 \ \& \ w_2! \, p_2}$	$\dfrac{\vdash w_1 = w_2 \quad \vdash p_1 = p_2}{\vdash w_1! \, p_1 = w_2! \, p_2}$	$\dfrac{\vdash f_1 = f_2}{\vdash ?f_1 = ?f_2}$
Abstraction	$\dfrac{\forall v. \vdash p_1[v/x] = p_2[v/x]}{\vdash [x]p_1 = [x]p_2}$		
Conditional	$\vdash \text{if true then } p_1 \text{ else } p_2 = p_1$	$\vdash \text{if false then } p_1 \text{ else } p_2 = p_2$	
Axioms	$\vdash x? \, \mathbf{0} = \mathbf{0}$	$\vdash (x? \, w! \, p) \ \& \ (w! \, p) = w! \, p$	

Not shown are the rules for α-conversion of process abstractions, and for reflexivity, symmetry and transitivity of equalities.

Table 3. The Proof System

Equations derived by the proof system are of the form $\vdash p_1 = p_2$, $\vdash w_1 = w_2$ or $\vdash f_1 = f_2$ dealing with (closed) processes, data expressions and process abstractions respectively. That is, "=" is overloaded. The inference rules of the proof system are given in Table 3; not shown are rules for reflexivity, symmetry and transitivity of the "=" relations, and for α-conversion of abstractions. Inference rules for data equality are to be supplied by the user.

To motivate the rules, consider

$$x? \, (x \bmod 2) \, ! \, \mathbf{0} \sim x? \, \text{if odd } (x) \ \text{then } 1! \, \mathbf{0} \text{ else } 0! \, \mathbf{0}$$

13

To prove this, the third congruence rule is needed, followed by the abstraction and the conditional rules. But to prove

$$x? (x \bmod 2) \, ! \, 0 \sim x? (\text{if odd} \, (x) \ \text{then} \ 1 \ \text{else} \ 0) \, ! \, 0$$

(where the conditional is not part of process syntax, but part of the data language), only congruence rules are needed together with a proof that

$$x \bmod 2 = \text{if odd} \, (x) \ \text{then} \ 1 \ \text{else} \ 0$$

Proposition 13. *The inference rules of the proof system are sound w.r.t. $\sim$.*

Proof. The rules of equivalence and congruence reflect the corresponding statements about $\sim$, the abstraction rule is how $\sim$ over abstractions is defined. The conditional rules and axioms are easily checked.

Definition 14 Depth. The depth d of a finite guarded sum is given by

$$
\begin{aligned}
d\,(\emptyset) &= 0 \\
d\,(w! \, p) &= 1 + d\,(p) \\
d\,(?f) &= 1 + \max_v d\,(f \ v) \\
d\,(f \ \& \ s) &= \max\,(d\,(?f)\,, d\,(!s)) \\
d\,(\text{if } b \text{ then } p_1 \text{ else } p_2) &= \text{if } b \text{ then } d\,(p_1) \text{ else } d\,(p_2)
\end{aligned}
$$

An abstraction can only have a finite nesting of conditionals, so that even if α is infinite, $\max_v$ is well defined.

Proposition 15 Completeness. *If p_1, p_2 are finite guarded sums and $p_1 \sim p_2$ then $\vdash p_1 = p_2$.*

Proof. Proceed by induction on the sum of the depths of p_1 and p_2. In the base case, this is 0, so both p_1 and p_2 are either $\mathbf{0}$, or conditionals that resolve to $\mathbf{0}$. In the latter case, use the inference rule Conditional to get $\vdash p_1 = p_2$.

For the induction step, there are several cases.

Suppose p_1 is $\mathbf{0}$. Then p_2 cannot have an output branch. It has to be $x? \, p_2'$ or a conditional that resolves to it. In the latter case, use the Conditional rules as often as needed. Then $\mathbf{0} \xrightarrow{v?} \mathbf{0}$ is matched by $x? \, p_2' \xrightarrow{v?} p_2'[v/x]$, and $p_2'[v/x] \sim \mathbf{0}$. By induction, $\vdash p_2'[v/x] = \mathbf{0}$. Now apply the first axiom.

Suppose p_1 is $w_1 ! \, p_1'$. Then p_2 has to be of the form $w_2 ! \, p_2'$ or $x? \, q \ \& \ w_2 ! \, p_2'$ (or conditionals resolving to these). In both cases, it must be that $w_1 = w_2$ and that $p_1' \sim p_2'$. By induction $\vdash p_1' = p_2'$. In the first case, the proof is completed by congruence. In the second, $p_1 \xrightarrow{v?} p_1$ is matched by $p_2 \xrightarrow{v?} q[v/x]$. By induction, $\vdash p_1 = q[v/x]$. Apply abstraction and the second axiom.

The remaining cases are either simple or symmetric to previous cases.

6 Expansion and Decomposition

For this section, an extended calculus CBS_e is introduced where guarded sums have an *output tree*, a finite set of output branches instead of just one as in CBS. The syntax of CBS_e is given by

$$p ::= \ !s \ \bigg| \ f \ \& \ s \ \bigg| \ p \mid p \ \bigg| \ \phi p_\beta \ \bigg| \ \text{if } b \text{ then } p \text{ else } p \ \bigg| \ A \ d$$

where $f ::= [x]p$ and $s ::= \{\langle w_i, p_i \rangle \big| i \in I\}$, where I is a finite set. The semantics below is extended from Table 1; the common parts are not repeated.

Guarded	$f \ \& \ s \xrightarrow{w_i!} p_i$	$\langle w_i, p_i \rangle \in s$	$[x]p \ \& \ s \xrightarrow{v?} p[v/x]$
Sum	$!s \xrightarrow{w_i!} p_i$	$\langle w_i, p_i \rangle \in s$	$!s \xrightarrow{v?} \ !s$

Now $!\emptyset$ says nothing and loops on all input. It corresponds to $\mathbf{0}$ in CBS. Similarly, $f \ \& \ \emptyset$ corresponds to $?f$ in CBS. Writing just $\langle w, p \rangle$ instead of $\{\langle w, p \rangle\}$, it is easily seen that CBS is a subcalculus of CBS_e.

Bisimulation extends naturally to CBS_e. A complete axiomatisation is similar to that for CBS; the lone axiom now is $([x]!s) \ \& \ s \sim \ !s$.

CBS_e processes too have a unique response to each input. Again, p/w denotes the image of p under w (see Definition 5). The finite number of output branches in a sum are enough for the expansion theorem below, since parallel compositions are finite. For legibility, $\{\langle w_i, p_i \rangle \big| i \in I\}$ is often written $\{w_i! p_i \big| i \in I\}$.

Proposition 16 Expansion theorem.

$$p_0 \mid p_1 \sim x? \ (p_0/x \mid p_1/x) \ \& \ \{w! \ (p'_r \mid p_{1-r}/w) \ \big| \ p_r \xrightarrow{w!} p'_r \ and \ r = 0, 1\}$$

For example, the process $2! \ p \mid 7! \ q$ can be expanded into a guarded sum:

$$2! \ p \mid 7! \ q \sim x? \ (2! \ p \mid 7! \ q) \ \& \{2! \ (p \mid 7! \ q) \ , 7! \ (2! \ p \mid q)\}$$

Since CBS is a subcalculus of CBS_e, the latter is obviously more expressive in some sense, but the following theorem shows that for every term in CBS_e, there is a strongly bisimilar term within CBS.

Proposition 17 Decomposition theorem. *The process $[x]p' \& \{s_i \big| i \in I\} : Proc \ \alpha$ can be decomposed into parallel components as follows. Let the translators ϕ, $\psi : Proc \ \alpha \to Proc \ \langle \text{Bool}, \alpha \rangle$ be defined by the graphs*

$$\phi = \{x \uparrow \langle \text{true}, x \rangle, \ \ \langle \_, x \rangle \downarrow x, \}$$
$$\psi = \{x \uparrow \langle \text{false}, x \rangle, \ \ \langle \_, x \rangle \downarrow x, \}$$

Let $q : \langle \alpha_\tau, Proc \ \alpha \rangle \to Proc \ \langle \text{Bool}, \alpha \rangle$ and $f : Proc \ \alpha \to \langle \text{Bool}, \alpha \rangle \to Proc \ \langle \text{Bool}, \alpha \rangle$ be given by

$$q \ s \ \overset{\text{def}}{=} \ \psi \ (x? \ \mathbf{0} \ \& \ s)$$
$$f \ p' \ \overset{\text{def}}{=} \ [y]. \text{if fst } y = \text{true then } \phi \ (p'[\text{snd } y/x]) \ \text{else } \mathbf{0}$$

Then $[x]p' \ \& \ \{s_i \ \big| \ i \in I\} \sim \phi^R \ (? \ (f \ p') \mid \prod_{i \in I} (q \ s_i))$.

15

Proof. First notice that

$$\forall v\colon \alpha. \quad \phi^{R\uparrow}\left(\psi^{\uparrow}v\right) = \psi_{\downarrow}\left(\phi^{R}_{\downarrow}v\right) = v$$

$$\forall v\colon \alpha. \quad \phi^{R\uparrow}\left(\phi^{\uparrow}v\right) = \phi_{\downarrow}\left(\phi^{R}_{\downarrow}v\right) = v$$

In the second line, the first two terms are both $\phi_{\downarrow}\left(\phi^{\uparrow}v\right)$. It follows that

$$\forall p\colon Proc\ \alpha. \quad p \sim \phi^{R}\left(\phi p\right) \sim \phi^{R}\left(\psi p\right)$$

The result follows by bisimulation upto $\sim$.

The CCS + laws can be applied to the output branches of CBS_e (they still make no sense for input), but they are now absorbed into the set syntax of the guarded sum.

7 Weak bisimulation

Let the function $\widehat{}$ be given by $\widehat{\tau !} = \varepsilon$ (the empty sequence) and $\widehat{w\natural} = w\natural$ if $w \neq \tau$.

Definition 18 Weak bisimulation for closed processes. $\mathcal{R} \subseteq \mathbf{P}_{cl} \times \mathbf{P}_{cl}$ is a weak bisimulation if whenever $p\mathcal{R}q$,

(i) if $p \xrightarrow{w\natural} p'$ then $\exists q'$ such that $q \xrightarrow{\tau!^{*}\widehat{w\natural}\tau!^{*}} q'$ and $p'\mathcal{R}q'$,

(ii) if $q \xrightarrow{w\natural} q'$ then $\exists p'$ such that $p \xrightarrow{\tau!^{*}\widehat{w\natural}\tau!^{*}} p'$ and $p'\mathcal{R}q'$

The largest weak bisimulation is an equivalence denoted $\approx$. It can be extended to input abstractions. The property below holds because sums are guarded.

Proposition 19. $\approx$ *is a congruence for CBS.*

Weak bisimulation here is formally similar to its CCS counterpart, but the effects are different. Firstly, $\tau! p \not\approx p$ in general! Let $p \stackrel{\text{def}}{=} x?\, x!\, \mathbf{0}$ and $q \stackrel{\text{def}}{=} \tau! p$. Then $p \not\approx q$, since p will always echo its input, but q may fail to do so: $q \xrightarrow{5?} q$. This cannot be matched by p since it has to receive, and become $5! \mathbf{0} \not\approx q$.

Despite these differences, the definition of $\approx$ is motivated as in CCS, because $\tau!$'s are autonomous and silent. That $\approx$ is in fact an observational equivalence would be established by characterising it as a testing equivalence [dNH84, Abr87]. This has not yet been done, but the present definition was arrived at via testing examples [Pra93a], and is consistent with them.

Proposition 20 Weak bisimulation laws.

1. $\tau!\, \mathbf{0} \approx \mathbf{0}$
2. $\tau!\, w!\, p \approx w!\, p$
3. $x?\, p\, \&\, \tau!\, x?\, p \approx x?\, p$
4. $f\, \&\, \tau!\, (f\, \&\, s) \approx f\, \&\, s$

It is a conjecture that the above laws are enough for a complete axiomatisation of $\approx$ for finite guarded sums. Readers familiar with the corresponding completeness proof in [Mil89] will note that the saturation technique there will not apply because there is no $+$ in CBS. A special case that can be proved from the laws above is that all "silent guarded sums", those that only use τ for w in the syntax, are equal to $\mathbf{0}$.

Laws 1 and 2 give two cases of p for which $\tau! \, p \approx p$. Another is the recursive process $X \stackrel{\text{def}}{=} x? \, (f \, \& \, \tau! \, X) \, \& \, s$.

In CBS_e, Law 4 includes Law 3, and Laws 1 and 2 are both captured in $\tau! \, s \approx s$. Here, and whenever context disambiguates, $!s$ can be written just s for legibilty. CBS_e also has analogues of the second and third τ laws of CCS.

$$\tau! \, w! \, p \approx \{w! \, p, \tau! \, w! \, p\}$$
$$w! \, (s \cup \{\tau! \, p\}) \approx \{w! \, (s \cup \{\tau! \, p\}) \, , w! \, p\}$$

8 Static contexts

The traditional process calculus specification of a good communication link is that it behave like a buffer. An arguably more natural requirement is that each user should experience its environment (consisting of the link and the user at the other end) as behaving like the other user. A telephone system works if it gives each user the feeling that the other is in the same room; it is not primarily the point that it behaves like a buffer. Further, a toy phone system could be modelled in CBS as consisting only of a translator at each end, say from voice to electrical signals and back (see Example 5 below). There are now no processes to which to apply the traditional specification. Thus the new correctness requirement amounts to a new verification paradigm, of which this section is a first presentation.

Review Example 2 in Section 2 of users u_0 and u_1 communicating via a link M, and the following discussion of reversibility of translators. In those terms, $s = \phi_0 u_0 \mid M \mid \phi_1 u_1$ describes the system from the viewpoint of M. From the point of view of u_0, the rest of the system is $\phi_0^R \, (M \mid \phi_1 u_1)$ and the suggested new specification of a good communication link is that

$$\phi_0^R \, (M \mid \phi_1 u_1) \approx u_1$$

This paper does not formalise the relation between behaviour at different points in a system; it appears that this can be done by an operational semantics of contexts, under development. But the new paradigm seems intuitively clear, and the algebraic apparatus needed to work with it is simple.

Definition 21 Contexts. A *static context* (context for short) is an expression given by
$$C ::= \text{hole} \, \Big| \, C \mid p \, \Big| \, p \mid C \, \Big| \, \phi C$$
where p is a process.

17

A context can be informally described as a process with a hole in it, and the definition deals only with the important subclass of processes that have a static operator outermost. Let C, D and E range over contexts.

Definition 22 Filling a context. Let C, D be contexts and q be a process. Then $C[q]$ is a process and $C[D]$ a context, defined inductively by

$$
\begin{aligned}
\mathsf{hole}\quad &[q] = q & \mathsf{hole}\quad &[D] = D \\
(C \mid p)\ &[q] = (C[q]) \mid p & (C \mid p)\ &[D] = (C[D]) \mid p \\
(p \mid C)\ &[q] = p \mid (C[q]) & (p \mid C)\ &[D] = p \mid (C[D]) \\
(\phi C)\quad &[q] = \phi\,(C[q]) & (\phi C)\quad &[D] = \phi\,(C[D])
\end{aligned}
$$

A simple result by induction on the structure of C is that $(C[D])\,[E] = C[D[E]]$.

A context has a hole inside and an environment outside. The operation below turns the context inside out, making the hole the environment and vice-versa.

Definition 23 Reverse of a context. The *reverse* of a context C, denoted C^R, is the context $turn\ C$ hole, where $turn$ is defined inductively on C.

$$
\begin{aligned}
turn\ \mathsf{hole}\quad D &= D \\
turn\ (C \mid p)\ D &= turn\ C\ \ (D \mid p) \\
turn\ (p \mid C)\ D &= turn\ C\ \ (p \mid D) \\
turn\ (\phi C)\quad D &= turn\ C\ \ \left(\phi^R D\right)
\end{aligned}
$$

Example 4 Some contexts and their reverses.

$$
\begin{aligned}
(i) \qquad & \mathsf{hole}^R & &= \mathsf{hole} \\
(ii) \qquad & \left(\mathsf{hole} \mid q\right)^R & &= \mathsf{hole} \mid q \\
(iii) \qquad & \left(\phi\ \mathsf{hole}\right)^R & &= \phi^R \mathsf{hole} \\
(iv) \qquad & \left(r \mid \phi\ \mathsf{hole}\right)^R & &= \phi^R\,(r \mid \mathsf{hole}) \\
(v) \qquad & \left(r \mid \phi\,(\mathsf{hole} \mid q)\right)^R & &= \phi^R\,(r \mid \mathsf{hole}) \mid q
\end{aligned}
$$

In the system $e \mid \phi p$ the environment of p is $\phi^R e$. More generally now, in $e \mid C[p]$, the environment of p is $C^R[e]$. An important special case is where $e = \mathbf{0}$, that is, where $C[p]$ describes the whole system. Then p's environment is the process $C^R[\mathbf{0}]$.

Now $\left(\phi^R\right)^R = \phi$. The proposition below generalises this to contexts. The lemmas are proved by induction on C. The latter needs $(C[D])\,[E] = C[D[E]]$.

Lemma 24. $\left(turn\ C\ D\right)^R = turn\ D\ C$

Lemma 25. $C^R[D] = turn\ C\ \ D$

Proposition 26. $\left(C^R[D]\right)^R = D^R[C]$

Proof. $\left(C^R[D]\right)^R = \left(turn\ C\ D\right)^R = turn\ D\ C = D^R[C]$

18

Putting $D = \mathsf{hole}$ in the above proposition yields $(C^R)^R = C$. Note that $C[\mathsf{hole}] = C$. Two other simple results are

$$(C \mid p)^R [D] = turn\ (C \mid p)\ D = turn\ C\ (D \mid p) = C^R[D \mid p]$$

and

$$(\phi C)^R [D] = turn\ (\phi C)\ D = turn\ C\ (\phi^R D) = C^R[\phi^R D]$$

Example 5 A toy telephone. Suppose the signals carried by the phone system are modelled simply as tagged speech, making a system like that of Example 2, but without the M. Let

$$\psi_w = \{x \uparrow \langle w, x \rangle, \langle 1 - w, x \rangle \downarrow x\}$$

The difference between ϕ_w of Example 2 and ψ_w is that the former delivers to u_w speech tagged w, while the latter delivers speech tagged $1 - w$. Recall that by convention (see Table 2), the above specification of ψ means that speech tagged w is mapped to τ. Then

$$s = \psi_0 u_0 \mid \psi_1 u_1$$

is a communication system that involves no processes as media. But it does link the users, who are not in direct communication. It could be even simpler, using the same translator and tag at both ends. The slightly more informative system above gives a direction to the signals.

But is the link good? Because it is not a process, there is no way to talk about its behaviour. But let

$$C_0 = \psi_0\ (\mathsf{hole}) \mid \psi_1 u_1$$

Then $s = C_0[u_0]$, so that u_0's environment is $C_0^R[\mathbf{0}]$, and the following statement is easy to check.

$$C_0^R[\mathbf{0}] = \psi_0^R\ (\psi_1 u_1) \sim u_1$$

That is, u_0 cannot tell whether it is communicating directly with u_1, or via the communication link. The situation for u_1 is symmetric.

Example 6 A ping-pong protocol with relays. The process u below announces integers from a list, waiting for a response from the environment before each one.

$$
\begin{aligned}
u\ [\,] &= \mathbf{0} \\
u\ (n : ns) &= x?\ n!\ (u\ ns) \\
u_0 &= 0!\ (u\ [2, 4, 6, ..]) \\
u_1 &= u\ [1, 3, 5, ..] \\
t &= u_0 \mid u_1
\end{aligned}
$$

u_0 and u_1 trade integers in a ping-pong protocol:

$$t \xrightarrow{\ 0!\ } \xrightarrow{\ 1!\ } \xrightarrow{\ 2!\ } \cdots$$

19

But now suppose the users communicate via the link M below, with translators as in Example 2.

$$M \stackrel{\text{def}}{=} y? \langle (1 - \text{fst } y) , \text{snd } y \rangle! \, M$$
$$\phi_w \stackrel{\text{def}}{=} \{ x \uparrow \langle w, x \rangle, \langle w, x \rangle \downarrow x \}$$

Then the whole system s is $\phi_0 u_0 \mid M \mid \phi_1 u_1$. Let

$$C_0 = \phi_0 \, (\text{hole}) \mid M \mid \phi_1 u_1$$

Then $s = C_0[u_0]$, so that u_0's environment is $C_0^R[\mathbf{0}]$, and the system described from u_0's viewpoint is $s_0 = u_0 \mid C_0^R[\mathbf{0}]$.

$$C_0^R[\mathbf{0}] = \phi_0^R \, (M \mid \phi_1 u_1) \approx u_1 \qquad \text{and so} \qquad s_0 \approx t$$

There is a symmetric situation from u_1's viewpoint. It is typical that when the link involves processes, the equivalence involved is weak rather than strong. It also matters how the users expect to use the link, i.e., what protocol they use.

9 CBS in a functional framework

Process specification language. In a CBS process abstraction $[x]p$, or in a parameterised constant definition $p\,x$, the dependence of p on x is expressed ultimately either by w (if $p \equiv w!\,q$ or $p \equiv f \,\&\, w!\,q$) or by a boolean function of x (if p is a conditional). A natural generalisation is to allow p to be any function of x expressible in some chosen framework. This generalisation is used in the examples in the rest of this paper. In the terms $?f$ and $f \,\&\, s$, the examples express f in a functional language, essentially a simple subset of Haskell, but with the freedom to use ordinary mathematical notation when appropriate. The input rules are

$$?f \xrightarrow{v?} f \; v \qquad\qquad f \,\&\, s \xrightarrow{v?} f \; v$$

so that substitution of data into process expressions is replaced by function application from the framework, and conditionals are no longer process syntax, but merely a particular kind of function.

This paper does not put this generalised process specification on a firm footing; that is ongoing work. But the generalisation makes for a natural, concise and powerful language, and it is usually clear how to translate examples to formal CBS. For example, consider the case construct. If b is a boolean and n an integer, then pairs $\langle b, n \rangle$ are built by a pair *constructor*. Then

$$\begin{array}{l} \text{case } x \text{ of} \\ \quad \langle \text{true}, n \rangle \to 1 + n \\ \quad \langle \text{false}, n \rangle \to 2 + n \end{array}$$

is a function that takes $\langle \text{true}, 5 \rangle$ to 6, and $\langle \text{false}, 6 \rangle$ to 8, and so on. It can be expressed $[x]$if fst x then $(1 + \text{snd } x)$ else $(2 + \text{snd } x)$. Thus the *pattern matching* provided by case makes *destructors* unnecessary (functions such as fst and snd that take elements of a structured data type and return components).

Constant definitions are also usually expressed by cases. Case analysis is sometimes qualified by a boolean condition. See the example below.

9.1 Examples

Example 7 Milner's scheduler. Processes p_i, $i \in 1..n$, each perform a task repeatedly, and are to be scheduled cyclically by signals go_i. The end of each task is signalled by $done_i$. The specification of the scheduler is $\phi\left(s\left(1, \emptyset\right)\right)$, where

$$\phi = \left\{ go_i \uparrow go_i, done_i \downarrow done_i \mid i = 1..n \right\}$$

$$s\left(i, X\right) \mid i \in X = x? \text{ case } x \text{ of}$$
$$done_j \to s\left(i, X - \{j\}\right)$$
$$s\left(i, X\right) \mid i \notin X = x? \text{ case } x \text{ of}$$
$$done_j \to s\left(i, X - \{j\}\right)$$
$$\& go_i! \left(s\left(i + 1, X \cup \{i\}\right)\right)$$

Here i says whose turn it is, and X is the set of active processes; $i + 1$ and $i - 1$ are calculated modulo n. This is close to Milner's specification [Mil89]. ϕ restricts incoming go_i's and the missing case in the incomplete case analyses above will not occur. Without ϕ, the scheduler has to explicitly ignore go_i's.

The scheduler can be implemented as s' below, a set of cells which schedule their respective wards and then wait for $done_i$ and go_{i-1} to happen in either order. To start with, only $\psi_1 a$ is ready to schedule its ward; the others wait for scheduling signals. Since no processes are active as yet, there cannot be any termination signals.

$$a = go!\, b$$
$$b = x? \text{ case } x \text{ of}$$
$$done \to d$$
$$go \to c$$
$$c = done?\, a$$
$$d = go?\, a$$
$$s' = \psi_1 a \mid \prod_{i \neq 1} \psi_i d$$
$$\psi_i = \{ go \uparrow go_i, go_{i-1} \downarrow go, done_i \downarrow done \}$$

Since the go_i can be heard both by p_i and by b_i or d_i, there is no need to relay the information by new signals. The following relation is a strong bisimulation upto $\sim$, and so $\phi s' \sim \phi\left(s\left(1, \emptyset\right)\right)$. The example suggests work on the notion "equivalent in a context", studied for CCS in [Lar87].

$$\left\{ \left\langle \phi\left(s\left(i, X\right)\right), \phi\left(\psi_i c \mid \prod_{j \in X, j \neq i} \psi_j b \mid \prod_{j \notin X} \psi_j d\right) \right\rangle \mid i \in X \right\}$$
$$\cup \left\{ \left\langle \phi\left(s\left(i, X\right)\right), \phi\left(\psi_i a \mid \prod_{j \notin X, j \neq i} \psi_j d \mid \prod_{j \in X} \psi_j b\right) \right\rangle \mid i \notin X \right\}$$

In a communication model that has to relay the go_i's, the bisimulation would be weak.

Example 8 Broadcast sort. The process *sorter* below listens for a list of integers, assumed to be all distinct. The more general case needs a little more detail but is almost as easy.

$$sorter = in\,(-, \top)$$
$$in\,(l, u) = x?\ \text{if } x = \top \text{ then } out\,(l, u) \text{ else}$$
$$\qquad\qquad\quad \text{if } l < x\ \ and\ \ x \leq u \text{ then } in\,(l, x) \mid in\,(x, u)\ \text{ else}$$
$$\qquad\qquad\quad in\,(l, u)$$
$$out\,(-, \top) = \mathbf{0}$$
$$out\,(-, u) = u!\,\mathbf{0}$$
$$out\,(l, u) = l?\,u!\,\mathbf{0}$$

Broadcast sort is a parallelised insertion sort. The input so far is held in a sorted list, maintained by cells each holding a number u and a "link" l, the next lower number. Let $-$ and $\top$ be sentinel values, respectively less than and greater than all numbers. There is always exactly one cell with $l = -$, and exactly one with $u = \top$. The next input number splits exactly one cell into two. At the end of input, marked by $\top$, output is initiated by the cell with $-$ announcing its u. Each cell (l, u) outputs u when it hears l.

A proof of correctness is the following bisimilarity with a specification in terms of a sorting function *sort* and lists xs of numbers. $\top$ is the end marker for the input list. The "cons" operator is written as an infix ":".

$$s\ xs = x?\,\text{if } x = \top \text{ then } h\ xs \text{ else } s\,(sort\,(x:xs))$$
$$h\ [\,] = \mathbf{0}$$
$$h\,(x:xs) = x!\,(h\ xs)$$

Its behaviour is illustrated below. The sorting function is used to do an insertion sort as each input element comes in.

$$s\ [\,] \xrightarrow{\ 5?\ } s\ [5] \xrightarrow{\ 8?\ } s\ [5, 8] \xrightarrow{\ 7?\ } s\ [5, 7, 8] \xrightarrow{\ \top?\ }$$
$$h\ [5, 7, 8] \xrightarrow{\ 5!\ } h\ [7, 8] \xrightarrow{\ 7!\ } h\ [8] \xrightarrow{\ 8!\ } \mathbf{0}$$

Let $n \geq 1$ and suppose the numbers $x_1, \ldots, x_n$ are all distinct.

$$insys\ [\,] \qquad\qquad\quad = in\,(-, \top)$$
$$insys\ [x_1, \ldots, x_n] \ = in\,(-, x_1) \mid in\,(x_1, x_2) \mid \ldots \mid in\,(x_n, \top)$$
$$outsys\ [\,] \qquad\qquad\ = out\,(-, \top)$$
$$outsys\ [x_1, \ldots, x_n] = out\,(-, x_1) \mid out\,(x_1, x_2) \mid \ldots \mid out\,(x_n, \top)$$

Then the relation

$$\{\ \langle s\ xs, insys\,(sort\ xs)\rangle,$$
$$\langle h\ xs, outsys\,(sort\ xs)\rangle\ \Big|\ xs \text{ any list of distinct numbers}\}$$

is a strong bisimulation. It proves that $s\ [\,] \sim sorter$.

Two disciplines that are generally followed in this paper are that the memory needed by any process, and the size of the spoken values, are both bounded. This allows the number of messages to be used as a measure of time taken, and the number of processes as a measure of space. The sorter is linear in the size of the input list, for both time and space.

Example 9 The alternating bit protocol without a medium. The process u_0 in the ping-pong protocol of Example 6 says 0 and then waits for a response from u_1 before saying 2. If it does anything else when it should be waiting, such as fetching the 2 from another process, it could miss u_1's response. An alternating bit protocol can synchronise the two users even if they miss a response as above. In the system below, the users use pairs of booleans and integers to try to send the integers to each other (the boolean being the alternating bit), and then report the integers they get from each other. The notation $\bar{b}$ stands for "not b".

$$u_w\ b\ n = x?\ (\ \text{case}\ x\ \text{of}$$
$$\langle b', n'\rangle\ \Big|\ w = 0 \wedge \bar{b} = b' \to n'!\ \big(u_w\ \bar{b}\ (n+2)\big)$$
$$\langle b', n'\rangle\ \Big|\ w = 1 \wedge b = b' \to n'!\ \big(u_w\ \bar{b}\ (n+2)\big)$$
$$\_ \qquad\qquad\qquad \to u_w\ b\ n$$
$$)$$
$$\&\langle b, n\rangle!\ (u_w\ b\ n)$$

Thus u_0 true 0 will repeat $\langle\text{true}, 0\rangle$ until it hears a $\langle\text{false}, n'\rangle$ when it will say n' and advance to u_0 false 2. While it is trying to say n', the user is deaf. Similarly u_1 true -1 will repeat $\langle\text{true}, -1\rangle$ until it hears a $\langle\text{true}, n'\rangle$ when it becomes deaf till it says n' and advances to u_1 false 1.

Let
$$s \overset{\text{def}}{=} \phi\ (u_0\ \text{true}\ 0 \mid u_1\ \text{true}\ -1)$$

where $\phi = \{n \uparrow n\}$ restricts all input and hides the pairs. Note that u_1 begins with a dummy -1 ignored by u_0 because it has the wrong alternating bit. Then $s \approx 0!\,1!\,2!\,3!\ldots$ is easy to show. It is also easy to extend the code to include (lossy) media, to include end users, to wrap up subsystems in translators to get more structure, and to specify correctness using context reversal.

This protocol is traditionally used to overcome a lossy medium; see for example [Mil89]. Users wait for a while for input and then autonomously retransmit. In CBS, lossy transmission can occur without a lossy medium. Also, waiting for input and autonomously speaking if not preempted is exactly what a guarded sum does. Hence the simplicity of the program above, which nonetheless is the heart of the usual protocol: the behaviour is similar.

10 A CBS simulator.

"Step". A simulator for CBS allows the user to run processes, in the sense discussed in Section 2, by means of the function

$$step\colon Proc\ \alpha \to OracleTree \to Result\ \alpha$$

where
$$result\colon\colon = Refuses\ \Big|\ Says\ w\ p$$

defines the type $Result\ \alpha$. Here $w\colon\alpha_\tau$ and $p\colon Proc\ \alpha$, as usual.

The basic idea is that $step\ (3!\,p) = Says\ 3\ p$, and $step\ (x?\,p) = Refuses$, but if $r \stackrel{\text{def}}{=} 3!\,p \mid 5!\,q$, should $step\ r$ be $Says\ 3\ \ (p \mid 5!\,q)$ or $Says\ 5\ \ (3!\,p \mid q)$?

$$r \stackrel{3!}{\longrightarrow} p \mid 5!\,q \qquad\qquad\qquad r \stackrel{5!}{\longrightarrow} 3!\,p \mid q$$

One way [Bur88] to achieve nondeterminism with functions is to put the nondeterminism in the data. $step$ is given an extra boolean argument, an *oracle*, whose value will be determined at run time, but once fixed will not change. The oracle chooses between parallel components if they compete; otherwise it has no effect. Thus for some oracle trees ot and ot',

$$step\ r\ ot\ = Says\ 3\ \ (p \mid 5!\,q)$$
$$step\ r\ ot'\ = Says\ 5\ \ (3!\,p \mid q)$$

$step$ needs a tree of oracles rather than a single oracle, because after choosing the right branch in $p \mid (q \mid r)$, a further choice has to be made.

A simulator is correct if it guarantees the following.

Definition 27 Implementation Requirement.

$(i)\ \ p \stackrel{w!}{\longrightarrow} p'\ \ \text{iff } \exists ot.\, step\ p\ ot = Says\ w\ p'$

$(ii)\ \forall w.\, p \stackrel{w!}{\longrightarrow}\!\!\!\!/\ \text{iff } \forall ot.\, step\ p\ ot = Refuses$

"Run". The function $trun : Proc\ \alpha \to [Oracle\,Tree] \to [\alpha_\tau]$ takes a process p and a list ots of oracletrees, and produces a run of p. It is defined in terms of $step$:

$$
\begin{aligned}
trun\ \ p\ \ (ot\!:\!ots) = \ &\textsf{case } step\ p\ ot\ \textsf{ of} \\
&Says\ w\ p'\ \ \to w\!:\!(trun\ \ p'\ ots) \\
&\underline{\ \ }\qquad\qquad \to [\ ]
\end{aligned}
$$

A correct simulator will therefore guarantee the following:

Definition 28 Corollary to Implementation Requirement.

$(i)\ \ p \stackrel{w!}{\longrightarrow} p'\ \ \text{iff } \forall ots.\, \exists ot.\, trun\ \ p\ \ (ot\!:\!ots) = w\!:\!trun\ p'\,ots$

$(ii)\ \forall w.\, p \stackrel{w!}{\longrightarrow}\!\!\!\!/\ \text{iff } \forall ots.\, trun\ \ p\ \ ots = [\ ]$

The function run produces an audible run. It is defined by $run\ \ p\ \ ots = striptau\ \ (trun\ \ p\ ots)$, where $striptau$ filters out τ's from a list of α_τ's.

"Test". Another common interface function is

$$test : Proc\ \alpha \to Proc\ \alpha \to [Oracle\,Tree] \to [\alpha]$$

Here $test\ p\ t\ ots$ puts p and t in communication, but returns only whatever p says. It is as though the user were in the same room as p when it is on the phone to t.

$$
\begin{aligned}
test\ p\ t\ ots &= run\ \ (\phi\ (\psi_p p \mid \psi_t t))\ \ ots \\
\psi_p &= \{w \uparrow w^p, w^t \downarrow w\} \\
\psi_t &= \{w \uparrow w^t, w^p \downarrow w\} \\
\phi &= \{w^p \uparrow w\}
\end{aligned}
$$

The translators flag the utterances of p and t differently, and pass only the former on to the outside. $test$ frequently allows a simple data type where run would need separate constructors to distinguish p's utterances from t's.

10.1   Examples

All preceding examples can be run on the simulator. Those below use *run* or *test* explicitly.

Example 10.
$$\phi \qquad\quad = \{n \uparrow n \mid odd\ n\}$$
$$s\ [\,] \qquad = \mathbf{0}$$
$$s\ (x\!:\!xs) = x!\,(s\ xs)$$

Then *run* $(\phi\,(s\ l))$ *ots* = *filter odd l* for any *ots*.

Example 11 Maximum of a list. Below, *cell n* announces *n* except if it hears a larger value first. So *cells l* announces an increasing sequence. The largest value in *l* is the last element of *maxrun l ots*. The algorithm takes linear time to create *cells*, but the number of announcements is in general less than the size of *l*. Correctness is best proved by an invariant.

$$cell\ n \qquad\quad = x?\,\text{if } x \geq n \text{ then } \mathbf{0} \text{ else } cell\ n$$
$$\qquad\qquad\qquad \& n!\,\mathbf{0}$$
$$cells\ l \qquad\quad = \textstyle\prod\,(map\ cell\ l)$$
$$maxrun\ l\ ots = run\quad (cells\ l)\ \ ots$$

The program illustrates an important aspect of CBS, that termination detection (when all component processes have fallen silent) cannot be done within CBS. It has to be done by the simulator, by returning a finite output stream.

Example 12 Fibonacci numbers.

$$add\ n = n!\,v?\,(add\ (n+v))$$
$$buf\ n = m?\,n!\,(buf\ m)$$
$$fib\ ots = test\ (0!\,(add\ 1))\,(v?\,(buf\ v))\ \ ots$$

The program is deterministic (all *ots* produce the same result).

10.2   Implementation of the simulator

The main code for a simple big-step simulator is given below. It should be self-explanatory. The simulator has been proved correct; this fact is used in ongoing work to prove executable CBS programs. [Gim95] is independent work on similar proofs and uses the simulator below.

$$step\ p\ (Node\ b\ lt\ rt) =$$

```
     case p of
          0           →  Refuses
          w!p         →  Says  w p
          ?f          →  Refuses
          ?f & w!p    →  Says  w p
          p | q       →  if b then pref  p  q  lt else pref  q  p  rt
          φp          →  case step p (Node b lt rt)
                            Says  u  p' → Says  (φ↑ u)  (φ  p')
                            Refuses → Refuses
```

25

$$pref : Proc\ \alpha \rightarrow Proc\ \alpha \rightarrow Oracle\, Tree \rightarrow Result\ \alpha$$
$$pref\ p\ q\ t\ =$$

```
        case step p t of
            Refuses    → case step q t of
                              Refuses    → Refuses
                              Says w q′ → Says w  ((p/w) | q′)
            Says w p′ → Says w  (p′ | (q/w))
```

The oracle tree, analysed into a node b and subtrees lt and rt, is needed only in $step\ (p\mid q)$, where b decides which of p or q to prefer. The other is examined only if the preferred one has nothing to say.

Implementation notes. Implementations in practical use are more sophisticated, cleaning up **0**'s in parallel compositions, and implementing $p_1 \mid p_2 \mid \ldots \mid p_n$ as a flat structure rather than a tree. In the sequential implementation above, the "oracle" is just a boolean, giving preference to one of the components. A parallel implementation would evaluate $step\ p\ lt$ and $step\ q\ rt$ in parallel and use the oracle to record which evaluates faster. If the host language has no existential types (see Section 3), the simulator implements process constructors as functions that yield an extended sum as in CBS_e, which can then be run. Finally, note the polymorphic recursion in $step$ for the translation case.

Divergent evaluation. Let $-$ be a divergent value. Then $step\ (3!\,p \mid -!\,q)\ ot$ gets stuck or yields $Says\ 3\ (p \mid (-!\,q)/3)$ depending on ot. Thus the simulator allows $?\,f\,\&\,w!\,p$ to be rescued by hearing if w is divergent. Simultaneity here is virtual, so that speech actually happens before hearing. In a parallel implementation, faster components do not have to wait for slower ones to finish computing.

Not even $3!\,p \mid -$ necessarily stops communication. This is reasonable for distributed systems. The implementation thus behaves as if $- \xrightarrow{w?} -$, i.e., as if $- \sim \mathbf{0}$. This judgement could be added to CBS. Diverging evaluations have been ignored in this paper for simplicity, but are a topic of ongoing work.

Quasi-parallel implementation. CBS has been implemented on top of a quasi-parallel evaluator [RW93], conveying the effect that each process runs on a separate processor. The evaluated language here is Haskell augmented with parallel evaluation annotations. These are notoriously difficult to program with. Thus CBS becomes a high level annotation language for parallel evaluation.

Some of the examples in this paper are typically "concurrent", such as protocols, and some are "parallel", such as the sorter. Profiles from the evaluator show that most of the programs here are so fine grain that communication overheads drown the parallelism, at least in this implementation. The CBS formulation is then valuable for its structure rather than parallelism. By contrast, a distributed search, not shown here since it makes more sense in Timed CBS, allows significant parallelism and meaningful experiments in optimisation and load balancing by varying the number of processors and the grain of searching.

11   Discussion

11.1   CBS with a + operator

Suppose an operator "+" is added to CBS, defined by the usual rules

$$\frac{p \xrightarrow{w\natural} p'}{p + q \xrightarrow{w\natural} p'} \qquad\qquad \frac{p \xrightarrow{w\natural} p'}{q + p \xrightarrow{w\natural} p'}$$

Then if $p \xrightarrow{w?} p'$ and $q \xrightarrow{w?} q'$, it follows that $p + q \xrightarrow{w?} p'$ and $p + q \xrightarrow{w?} q'$, i.e., + is always resolved non-deterministically on any input. Worse, if $\phi_{\downarrow} v = \tau$, then $\phi p + \phi q \xrightarrow{v?} \phi p$ and $\phi p + \phi q \xrightarrow{v?} \phi q$, so the + is resolved by a lost value.

[Pra93a] presents a version of CBS with a + operator (called "CBS$^+$" below) that resolves sensibly. In CBS$^+$, losses $\mathbf{0} \xrightarrow{v:} \mathbf{0}$, $w!p \xrightarrow{v:} w!p$ and $p \xrightarrow{\tau:} p$ are distinguished from hearing, the two being mutually exclusive. The problem with + is solved by a rule that says $p + q$ loses a value only if both p and q do. Parallel composition extends naturally by $! \bullet : = !$, $? \bullet : = ?$ and $: \bullet : = :$.

Observational equivalence in CBS$^+$. $\sim$ distinguishes between $w?$ and $w:$ but $\approx$ is defined not to. Thus $x?\,\mathbf{0} \not\sim \mathbf{0}$ but $x?\,\mathbf{0} \approx \mathbf{0}$. The latter is observational, since the environment cannot tell whether a process accepted or lost input. The process $v?\,p$ can be defined operationally to lose all values except v, under which it evolves to p. Then

$$3?\,p \approx X \text{ where } X \stackrel{\mathrm{def}}{=} x?\,\textsf{if } x = 3 \textsf{ then } p \textsf{ else } X$$

But $3?\,p + 4?\,q \xrightarrow{5:} 3?\,p + 4?\,q$, while $X + 4?\,p \xrightarrow{5?} X$. That is, + does not respect $\approx$, which is therefore not a congruence.

Axiomatisations. For CBS$^+$, the usual + laws of commutativity, associativity, idempotence and $\mathbf{0}$-identity hold. They constitute a complete axiom system for strong bisimulation, as they do for CCS; this reflects the fact that strong bisimulation ignores the communication model. Indeed for pure CBS$^+$, the calculus with only $v?\,p$ and no $x?\,p$, the proof is identical to that of [Mil89].

Recent independent work [HR95] has also axiomatised weak barbed bisimulation for CBS$^+$. As might be expected, the + plays a crucial role.

CBS$^+$ versus CBS. + is not derivable [Pra87, Pra89] from CBS, since $\approx$ is a congruence for CBS and + does not respect it. Thus CBS$^+$ is strictly more expressive. (In CBS, $3?\,p$ is *defined* as the X above; the two cannot be distinguished). However, examples suggest that CBS has the same programming power as CBS$^+$. CBS is more compact than CBS$^+$, uses standard definitions of bisimulation, and in the laws $x?\,\mathbf{0} = \mathbf{0}$ and $(x?\,w!p) \,\&\, (w!p) = w!p$ has $\sim$ instead of $\approx$ for "=".

	CCS	CBS
Communication	Inaudible to bystanders	Audible to all
Transmission	Controlled	Autonomous
Input=Output?	In pure CCS	Not even in pure CBS$^+$
Silent actions	Only one, τ.	5?, 6?, and τ! are different silent actions.
Abstraction	From τ. Because it is internal? autonomous? silent?	From silent actions. Clearly seen in CBS$^+$.
Importance of τ	Central. Any interesting computation involves τ.	Any interesting computation involves autonomous actions, usually not τ!.
Interleaving	Because an observer can hear only one thing at a time.	Because of the single channel. Same order for all observers.
Scoping	Separate translation and restriction operators.	Restriction and hiding are special cases of translation.
Typing	Channels can be typed. Processes?	Processes are typed, with translators between different types.
Contexts	No obvious reversal.	Can be reversed.
Simulation	By interacting with the process.	By running it.

Table 4. Some differences between CCS and CBS

11.2   Comparison with CCS

Processes in CCS communicate on channels. Here $\bar{a}\,(5)$ is the action of sending the integer message "5" on channel a, and $a\,(5)$ that of receiving "5" on a. These wait for each other. Doing $\bar{a}\,(5)$ means discovering that the environment did $a\,(5)$, and vice-versa. Both a and $\bar{a}$ are controlled and audible. The rules for $|$ are

$$\frac{p \xrightarrow{\bar{a}(v)} p' \quad q \xrightarrow{a(v)} q'}{p \mid q \xrightarrow{\tau} p' \mid q'} \qquad\qquad \frac{p \xrightarrow{\bar{a}(v)} p'}{p \mid q \xrightarrow{\bar{a}(v)} p' \mid q}$$

with three more interleaving rules (for input by p, and both actions by q). The first rule is the communication rule; the action τ is autonomous and silent. The process $\mathbf{0}$ has no actions.

A striking special case occurs when the data type used is the unit type with only one element. In this *pure* CCS, used in most studies, input actions can be written just a, b, ... while $\bar{a}$, $\bar{b}$, ... will do for output actions. Communication here is pure synchronisation. It is usual to say that $\bar{\bar{a}} = a$, making it clear that calling an action "input" or "output" is pure convention.

For a summary comparison between CCS and CBS, see Table 4.

One-to-many communication. It might appear that the main difference between CCS and CBS is one-to-one communication versus one-to-many. This factor is

algorithmically important because one broadcast may sometimes achieve a goal that needs several relay transmissions in one-to-one models. But CBS differs semantically from CCS even when the one-to-many aspect of broadcast is not apparent, because of the input/output distinction.

Synchrony and asynchrony. These terms do not help classify CBS. According to [BKT84], a calculus has synchronous cooperation if every process has to act at every step, and synchronous communication if actions communicate only if performed simultaneously. Then cooperation is asynchronous and communication synchronous in CCS, while both are synchronous in CBS. But [SNP87] argue that broadcast communication is asynchronous since the sender of a message does not wait for the receiver.

What is clear is that CBS makes more distinctions than some calculi labelled "asynchronous". Those in [JJH90, JU90] have the results $a!\,b!\,P \approx b!\,a!\,P$ and $a?\,b?\,P \approx b?\,a?\,P$, neither of which hold in CBS.

Running processes. Since the only autonomous action in CCS is silent, running a process in the sense of this paper yields no information. Interaction can be done in "batch" mode with an environment that is then unpacked to find out what happened, but this goes against an extensional view of the environment as a process. Most tools for CCS are for reasoning about processes, not for simulation. There will presumably be more interest in programming *in* CCS with the emergence of several combinations of CCS-style concurrency with ML, including PFL [Hol83], Facile [GMP89], LCS [BS94] and CML [Rep91, BMT92].

Late and early semantics. Readers familiar with the "late" semantics convenient for value passing languages (see for example [HL95]) should note that as a consequence of input determinism, late and early bisimulations coincide in CBS.

11.3 Design decisions

Negative premises. In CBS$^+$ losses are mutually exclusive with hearing. That is, a loss encodes a negative premise (see also Harel's operator below). It is then sufficient as a simple stratification strategy [Gro90] that these can be derived independently of positive premises. A similar structure shows up in CBS as Proposition 8. Thus it is possible to formulate CBS too in terms of negative premises.

More parallelism? $\tau!\bullet\tau! = \tau!$ is the extent to which the single channel assumption can be relaxed to allow subsystems to proceed independently, gaining parallelism. Then the distribution law (see Proposition 10) for translation over $|$ is no longer a strong bisimulation, but a weak one. For suppose $\phi = \{n \downarrow n\}$, hiding all output. Then $\phi\,(5!\,\mathbf{0} \mid 6!\,\mathbf{0})$ still needs two steps to move to $\mathbf{0}$, but $\phi\,(5!\,\mathbf{0}) \mid \phi\,(6!\,\mathbf{0})$ can do so in one.

But further putting $v!\bullet\tau! = v!$ destroys input determinism, for the process doing the τ also "heard" v. Another effect is $\phi\,(5?\,p \mid 6!\,q) \mid 5!\,r \xrightarrow{5!} \phi\,(5?\,p \mid q) \mid r$. That is, $5?\,p$ heard the 6 and missed the overlapping 5.

Silent τ*'s.* An equivalent calculus results by removing the rule $p \xrightarrow{\tau?} p$, and adding

$$\frac{p \xrightarrow{\tau!} p'}{p \mid q \xrightarrow{\tau!} p' \mid q} \qquad \frac{q \xrightarrow{\tau!} q'}{p \mid q \xrightarrow{\tau!} p \mid q'} \qquad \phi p \xrightarrow{v?} \phi p \qquad \text{if } \phi_{\downarrow} v = \tau$$

This gives more efficient implementations.

11.4 Related work

Semantics. Work related only distantly to CBS includes an informal study [Geh84], and denotational semantics for broadcasting [SNP87, Bro88]. LINDA [CG89, KH94] is in effect a programming model using buffered broadcast, where receivers can read messages any time after they have been transmitted. Statecharts [Har87] and ESTEREL [BG92] both use unbuffered broadcast, but the communication models are rather different. ESTEREL, for example, allows multiple signals to be broadcast simultaneously, and the receiver chooses which to act on.

The theory of input/output automata [LT87] is closer, particularly now that it is in a process algebraic formulation [Seg92], but this theory is still different enough that a detailed comparison needs time, and is yet to be carried out.

The "broadcast" operator below is due to Harel [Pnu85].

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \mid F \xrightarrow{a} E' \mid F'} \qquad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} \!\!\!\!\!/}{E \mid F \xrightarrow{a} E' \mid F} \qquad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} \!\!\!\!\!/}{F \mid E \xrightarrow{a} F \mid E'}$$

The first rule describes multiway synchronisation. The second and third rules permit interleaving, but the negative premise ensures that E can do a by itself only if F cannot do a. Despite these necessary features, Harel's operator cannot describe broadcast because actions are not divided into transmissions and receptions. Note the symmetric role played by the participants.

Value-passing CSP [Hen90] does distinguish transmitter from receiver. It has multiway synchronisation, and indeed a notion of broadcasting, but a strange one where speakers can synchronise but listeners cannot.

Programming. Examples are hard to find. Even literature that describes broadcast as a primitive [BC91] gives no examples of use. It is perhaps relevant that the computation model of CBS can be classified as "multiple instruction single data stream", a class usually regarded as empty. It turns out however, that some of the examples in this paper are re-inventions, reported earlier in [HT92, YLC90, DK86]. Programming with broadcasts is clearly a small and rather neglected field of research. It is a sobering thought that the reasons for this neglect, whatever they are, may also apply to CBS.

Let k, k_i and π_ϕ be natural numbers. Refer to Table 1 for types of other variables. The syntax of PCBS$_e$ is given by

$$p ::= \ !s \ \Big| \ f \ \& \ s \ \Big| \ p|p \ \Big| \ \phi p_\beta \ \Big| \ \text{if } b \text{ then } p \text{ else } p \ \Big| \ A \ d$$

where $f ::= [x]p$ and $s ::= \{\langle w_i, k_i, p_i \rangle \mid i \in I\}$, where I is a finite set. Translators ϕ are specified by triples $\langle \pi_\phi, \phi^\uparrow, \phi_\downarrow \rangle$.

The priority $\pi(s)$ of an output tree s is given by the rules

$$\begin{aligned}
\pi(\emptyset) &= \infty \\
\pi\left(\{\langle w_i, k_i, p_i \rangle \mid i \in I\}\right) &= \min_{i \in I} k_i
\end{aligned}$$

The priority $\pi(p)$ of a closed process p is given by the rules

$$\begin{aligned}
\pi(!s) &= \pi(s) \\
\pi(f \ \& \ s) &= \pi(s) \\
\pi(p|q) &= \min(\pi(p), \pi(q)) \\
\pi(\langle \pi_\phi, \phi^\uparrow, \phi_\downarrow \rangle p) &= \pi_\phi + \pi(p) \\
\pi(\text{if } b \text{ then } p_1 \text{ else } p_2) &= \text{if } b \text{ then } \pi(p_1) \text{ else } \pi(p_2) \\
\pi(A \ d) &= \pi(p[d/z]) \text{ where } A \ z \stackrel{\text{def}}{=} p
\end{aligned}$$

Tau	$p \xrightarrow{\tau?k} p \quad k \le \pi(p)$
Guarded Sum	$f \ \& \ s \xrightarrow{w_i!k_i} p_i \quad \langle w_i, k_i, p_i \rangle \in s \wedge k_i = \pi(s) \qquad [x]p \ \& \ s \xrightarrow{v?k} p[v/x] \quad k \le \pi(s)$ $!s \xrightarrow{w_i!k_i} p_i \quad \langle w_i, k_i, p_i \rangle \in s \wedge k_i = \pi(s) \qquad\qquad\qquad\ !s \xrightarrow{v?k} !s \quad k \le \pi(s)$
Translate	$\dfrac{p_\beta \xrightarrow{u!k} p'_\beta}{\phi p_\beta \xrightarrow{\phi^\uparrow u!(k+\pi_\phi)} \phi p'_\beta} \qquad\qquad \dfrac{p_\beta \xrightarrow{\phi_\downarrow w?(k \,\dot-\, \pi_\phi)} p'_\beta}{\phi p_\beta \xrightarrow{w?k} \phi p'_\beta}$

The rules for parallel composition, conditionals and constant definition are the same as for CBS (see Table 1), except that "$w\natural$" on the arrows is replaced by "$w\natural k$".

Table 5. The syntax of PCBS$_e$ and the semantics of closed processes

12 CBS with priorities

Priority is a powerful and important tool in many forms of distributed and concurrent computing. Examples include interrupt processing, and the ability to let one message overtake another in a communication system. This section introduces the calculus PCBS, which is CBS with priorities. PCBS is similar to CBS in types, syntax and semantics, so only the differences will be pointed out.

A PCBS process does not just say "5", it says so at a priority it chooses. Priorities are natural numbers, with 0 the highest. CBS is the special case of PCBS where the priority of every utterance is 0. The priority of a process is defined to be k if it has an utterance at priority k. There can be only one such k;

if a process has several components, only the most urgent will speak. A process with nothing to say is said to have priority "∞", lower than priority k for any natural number k.

A process of priority k will not hear speech of lower priority than k. Such speech therefore cannot take place. To formulate this, the action of hearing an utterance is annotated with the priority of the utterance. Saying and hearing w are written $\xrightarrow{w!k}$ and $\xrightarrow{w?k}$, where k is the priority of the action.

To avoid repetition, only PCBS_e (that is, CBS_e with priorities) is developed. The syntax and semantics are given in Table 5. Priorities are used materially only in the rules for translation and in the side conditions for guarded sums and Tau, so only these rules are shown. PCBS is easily recovered from PCBS_e; it consists of the cases where I is empty or a singleton in the guarded sums. Except if the distinction matters, "PCBS" will be used for both calculi.

The output tree $\{\langle w_i, k_i, p_i \rangle \mid i \in I\}$ is also written $\{w_i!_{k_i} p_i \mid i \in I\}$, and $!s$ is written s when context disambiguates. Further, the singleton set $\{w!_k p\}$ may be written just $w!_k p$.

The process $?f$ has nothing to say. It hears any message v from the environment, no matter at what priority it is said, and becomes the process $f\ v$. Note that the priority of the message is not input.

$$?f \xrightarrow{5?3} f\ 5 \qquad\qquad x? \left(f \ \& \ (x+1)\ !_3\, p\right) \xrightarrow{5?4} f \ \& \ 6!_3\, p$$

The process $f \ \& \ 6!_3\, p$ has priority 3; it wishes to say 6 at priority 3 and become p. If it hears v, said at priority 3 or higher, it evolves to $f\ v$.

$$f \ \& \ 6!_3\, p \xrightarrow{6!3} p \qquad\qquad f \ \& \ 6!_3\, p \xrightarrow{4?k} f\ 4 \ \text{if } k \leq 3$$

The process $5!_3\, p$ wishes to say 5 at priority 3; it hears, but ignores, anything at priority 3 or higher. It refuses even to hear speech at priority lower than 3.

$$5!_3\, p \xrightarrow{5!3} p \qquad\qquad 5!_3\, p \xrightarrow{v?k} 5!_3\, p \ \text{if } k \leq 3$$

Following the recipe at the bottom of Table 5, here is one case of the $|$ rule.

$$\frac{p \xrightarrow{w!k} p' \quad q \xrightarrow{w?k} q'}{p \mid q \xrightarrow{w!k} p' \mid q'}$$

Here it is worth noting that the priorities as well as the messages have to be the same in both premises. The priority of the conclusion is the same as that of the premises. So $5!_{k+1}\, r$ cannot speak when in parallel with $3!_k\, s$, which will refuse to hear a message at priority $k + 1$. Similarly, the environment is allowed to speak only if it does so at priority k or higher. PCBS is a more centralised system than CBS.

A translator ϕ is specified by a triple $\langle \pi_\phi, \phi^\uparrow, \phi_\downarrow \rangle$. The process ϕp says $\phi^\uparrow 5$ at priority $k + \pi_\phi$ if p says 5 at k, and hears 4 at k if p hears $\phi_\downarrow 4$ at $k \doteq \pi_\phi$. Here $n \doteq m$ is $n - m$ if $n \geq m$ and 0 otherwise. Thus a translator can deprioritise a subsystem. Note that hiding speech does not hide its priority.

The rules for conditionals and constants are identical to those of CBS (see Table 1) except that the PCBS actions have priority annotations. The priority of the premise is carried over to the conclusion.

Properties of the calculus. The propositions below are all proved by induction on guardedness. They confirm that PCBS behaves as intended.

Let $p \xrightarrow{w \updownarrow k}$ mean "$\exists\, p'$ such that $p \xrightarrow{w \updownarrow k} p'$", and let $p \xcancel{\xrightarrow{w \updownarrow k}}$ mean "$\nexists\, p'$ such that $p \xrightarrow{w \updownarrow k} p'$".

Proposition 29. $\pi(p) = k$ *and* $k \neq \infty$ *iff* $\exists w$ *such that* $p \xrightarrow{w!k}$.

Therefore, $p \xrightarrow{w!k}$ and $p \xrightarrow{w'!k'}$ implies $k = k'$, and $p \xcancel{\xrightarrow{w!k}}$ if $k \neq \pi(p)$. The first of these facts can be proved independently in an alternative formulation that gives $\xrightarrow{\tau?}$ transitions as axioms for guarded sums only (and infers them for other processes). Such a formulation uses only the priority of an output tree in the operational semantics, and the priority of a process is only an explanatory notion.

Proposition 30. $\forall p, w, k \leq \pi(p), \exists!\, p'$ *such that* $p \xrightarrow{w?k} p'$.

Corollary 31. $p \xrightarrow{w?k}$ *iff* $k \leq \pi(p)$.

Thus hearing can be read as permitting the environment to speak.

Definition 32. p/w, the image of p under w, is the p' such that $p \xrightarrow{w?0} p'$.

13 Bisimulations for PCBS

Let $\mathbf{P}$ now be the set of all PCBS processes, and $\mathbf{P}_{cl}$ the set of closed ones.

Definition 33 Strong bisimulation for closed processes. $\mathcal{R} \subseteq \mathbf{P}_{cl} \times \mathbf{P}_{cl}$ is a strong bisimulation if whenever $p\mathcal{R}q$,
(i) if $p \xrightarrow{w \updownarrow k} p'$ then $\exists q'$ such that $q \xrightarrow{w \updownarrow k} q'$ and $p'\mathcal{R}q'$,
(ii) if $q \xrightarrow{w \updownarrow k} q'$ then $\exists p'$ such that $p \xrightarrow{w \updownarrow k} p'$ and $p'\mathcal{R}q'$

The largest strong bisimulation is an equivalence, denoted $\sim$. It can be extended to process abstractions in the usual way.

Proposition 34. $\sim$ *is a congruence for PCBS.*

Proposition 35 Strong bisimulation laws.

1. *Let* $s = \{w_i!_{k_i}\, p_i \mid i \in I\}$ *and* $s' = s \cup \{w!_k\, p\}$ *where* $k > \min_{i \in I} k_i$. *Then* $f\ \&\ s \sim f\ \&\ s'$ *and* $!s \sim !s'$.
2. *Let* $s \stackrel{\text{def}}{=} \{w_i!_{k_i}\, p_i \mid i \in I\}$ *and* $\phi s \stackrel{\text{def}}{=} \{(\phi^{\uparrow} w_i)\,!_{k_i + \pi_\phi}\, \phi p_i \mid i \in I\}$
   (a) $\phi\,(!s) \sim !\,(\phi s)$
   (b) $\phi\,([x]p\ \&\ s) \sim x?\,(\text{if } \phi_{\downarrow} x = \tau \text{ then } \phi\,([x]p\ \&\ s) \text{ else } \phi p[\phi_{\downarrow} x/x])\ \&\ (\phi s)$
3. $\phi\,(\psi p) \sim (\phi \circ \psi)\, p$ *where* $\phi \circ \psi$ *is specified by* $\langle \pi_\phi + \pi_\psi, \phi^{\uparrow} \circ \psi^{\uparrow}, \psi_{\downarrow} \circ \phi_{\downarrow} \rangle$.

Only the laws that distinguish PCBS from ordinary CBS are given above. The others carry over in the evident way. For example, the law $([x]!s)\ \&\ s \sim !s$ holds. Together with Law 1, this is enough to axiomatise finite guarded sums for PCBS; the proof is almost identical to that for CBS. First use Law 1 to put guarded sums in a standard form by dropping outranked branches.

Proposition 36 Expansion theorem.

$$p_0 \mid p_1 \sim x? \, (p_0/x \mid p_1/x) \,\&\, \{w!_k \, (p'_r \mid p_{1-r}/w) \mid p_r \xrightarrow{w!k} p'_r \text{ and } r = 0, 1\}$$

The expansion theorem is identical to that of CBS except that ! is annotated. It seems to ignore priorities, because it includes even what happens if the lower priority component speaks.

$$2!_1 \, p \mid 7!_2 \, q \sim x? \, (2!_1 \, p \mid 7!_2 \, q) \,\&\, \{2!_1 \, (p \mid 7!_2 \, q) \,,\, 7!_2 \, (2!_1 \, p \mid q) \, \}$$
$$\sim x? \, (2!_1 \, p \mid 7!_2 \, q) \,\&\, \{2!_1 \, (p \mid 7!_2 \, q) \qquad\qquad\qquad \}$$

But Law 1 applied to the r.h.s. shows that all is well. The expansion theorem motivates the presence of outranked branches in the output tree.

Definition 37 Priority abstracted bisimulation. $\mathcal{R} \subseteq \mathbf{P}_{cl} \times \mathbf{P}_{cl}$ is a priority abstracted bisimulation if whenever $p\mathcal{R}q$,

(i) if $p \xrightarrow{w\natural k} p'$ then $\exists q'$, k' such that $q \xrightarrow{w\natural k'} q'$ and $p'\mathcal{R}q'$,

(ii) if $q \xrightarrow{w\natural k} q'$ then $\exists p'$, k' such that $p \xrightarrow{w\natural k'} p'$ and $p'\mathcal{R}q'$

The largest priority abstracted bisimulation is an equivalence, denoted $\simeq$. This is not a congruence, for $3!_1 \, p \simeq 3!_2 \, p$, yet $3!_1 \, p \mid 5!_1 \, q \xrightarrow{3!1}$, while $3!_2 \, p \mid 5!_1 \, q \xrightarrow{3!k}\!\!\!\!\!/$ for any k. Let $\simeq^c$ be the largest congruence contained in $\simeq$.

Proposition 38. $\sim = \simeq^c$

Proof. It is easy to see that $\sim \,\subseteq\, \simeq$, and $\sim$ is a congruence. For the other direction, let $p \simeq^c q$, and let $p \xrightarrow{w\natural k} p'$. Then $q \xrightarrow{w\natural k'} q'$, and $p' \simeq^c q'$. But $k' = k$, otherwise a $\mid$ context can be found that can distinguish p and q.

Definition 39 Weak bisimulation for closed processes. For convenience, let $p \xrightarrow{\tau!} p'$ stand for $\exists k$ such that $p \xrightarrow{\tau!k} p'$. Then $\mathcal{R} \subseteq \mathbf{P}_{cl} \times \mathbf{P}_{cl}$ is a weak bisimulation if whenever $p\mathcal{R}q$,

(i) if $p \xrightarrow{w\natural k} p'$ then $\exists q'$ such that $q \xrightarrow{\tau!^* \widehat{w\natural} \tau!^*} q'$ and $p'\mathcal{R}q'$,

(ii) if $q \xrightarrow{w\natural k} q'$ then $\exists p'$ such that $p \xrightarrow{\tau!^* \widehat{w\natural} \tau!^*} p'$ and $p'\mathcal{R}q'$

The largest weak bisimulation is an equivalence, denoted $\approx$.

$\tau!_{k'} \, v!_k \, p \approx v!_k \, p$ only if $k' \leq k$. If $k' > k$ the left hand process will tolerate speech at priority k', but the right won't. An alternative definition of bisimulation for PCBS prescribes that τ's preceding the matching action have to be of equal or higher priority. Interestingly, the apparently more liberal definition above captures the same effect. As for CBS,

Proposition 40. $\approx$ *is a congruence for PCBS.*

14   Discussion

Analogy with time. Suppose a process of priority k waits for k seconds before making its request to speak. Then it will be preempted by less patient processes. This analogy goes quite a long way, inspiring a construct *timeout* $5\,(x?\,p\,\&\,w!\,q)$ that waits for 5 seconds for input. After that, it will say w and become q. Timed CBS [Pra95], under development, is based on this idea.

This analogy motivates why there is a highest priority rather than a lowest. A related reason is that *step* has to choose some speaker of highest priority. It helps if this is 0, for then it may have to look no further. If there is no highest priority, all processes must be asked every time.

From CBS to PCBS. It is instructive to consider alternative designs for PCBS, starting from the same syntax, and the same interpretation: $3!_1\,p$ wishes to say "3" at priority 1. A first attempt could be to make the parallel rule

$$\frac{p \xrightarrow{w!} p' \quad q \xrightarrow{w?} q'}{p \mid q \xrightarrow{w!} p' \mid q'}\,\pi\,(p) \leq \pi\,(q)$$

This allows $3!_1\,p \mid 4!_2\,q \xrightarrow{3!} p \mid 4!_2\,q$ to be derived, but not $3!_1\,p \mid 4!_2\,q \xrightarrow{4!} 3!_1\,p \mid q$.

But strong bisimulation in this calculus would be priority abstracted, and not a congruence. To repair this, annotate speech with the speaker's priority: $3!_1\,p \xrightarrow{3!1} p$. Now $3!_1\,p \not\sim 3!_2\,p$. The annotation is only to help reasoning; it is not part of what is said. The PA announcer reads out the chosen message, not why it was chosen over others. Proposition 29 holds for this intermediate calculus. The side-condition $\pi\,(p) \leq \pi\,(q)$ is equivalent to $q \xrightarrow{w!k} \!\!\!\!/\,$ if $k < \pi\,(p)$. Despite this negative premise, the transition system is well defined, for the definition of $\pi\,(p)$ is independent of the transitions of p.

The final step, that of annotating hearing with the priority of the speech heard, is less important. It encodes process priority into hearing transitions, and therefore absorbs the side-condition of the parallel rule, and the implicit negative premise, into the second premise. Corollary 31 describes the new annotation.

Value priority. A very similar calculus results if a fixed priority function $\zeta\!:\!\alpha \to \mathbf{N}$ is associated with the spoken data type. Translation functions can now alter the priorities of actions. For the calculus to be well behaved, it is sufficient that translation should preserve the priority order strictly (i.e., be monotonic, and maintain inequalities), and that $\zeta\,(\phi_\downarrow \phi^\uparrow v) = \zeta v$.

CCS with priorities. Adding priorities to CCS [CH90, CW95] is difficult, involving two stage operational semantics and other complications. The result is nonetheless unsatisfactory. The root problem is that the handshake makes an autonomous action out of two controlled ones, which are assigned priorities.

[CH90] considers prioritised and unprioritised actions (the former written here with primes). An a priori semantics labels transitions with actions as usual.

In the second stage, prioritised actions are unconstrained, but unprioritised actions can only take place if not preempted by prioritised τ's—a negative premise. The resulting new axiom is $a \, . \, p + \tau' . \, q \sim \tau' . \, q$, the cognate of Law 1 of Proposition 35. But the result is not entirely satisfactory, for $a \, . \, p + b' . \, q \not\sim b' . \, q$. [CH90] says that $\sim$ here would be a useful possibility; it also points out that then such actions cannot be restricted! This is precisely the scenario of PCBS.

Defining weak bisimulation for this model is non-trivial [NCCC94]; τ's are abstracted from sequences of actions in a priority sensitive way. (Remember that τ's preceding a matching action in PCBS must be of equal or higher priority).

[CW95] presents a priority sum $+'$ similar to Occam's PRIALT; $a \, . \, p +' b \, . \, q$ can perform a b only if the environment will not do $\bar{a}$. Now it is not clear whether $\big((a \, . \, p +' \bar{b} \, . \, q) \mid (b \, . \, r +' \bar{a} \, . \, s) \big) \backslash \{a, b\}$ can perform a τ. Actions are therefore separated into input and output and the initial actions of a prisum are required to be inputs. This breaks the symmetry of pure CCS, and complicates the syntax.

The transition relation is parameterised by the output actions the environment can do, as is the definition of bisimulation. The expected negative premise for $p +' q$ turns up as a side condition that the actions p can accept, computed independently of the transition system, should not be offered by the environment. Lastly, a ready function is needed to adjust the environment parameter upon communication in a parallel composition.

It is almost as though [CW95] declared output actions autonomous and input actions controlled, and studied what could (autonomously) happen. Unfortunately for this interpretation, output actions can be restricted. Worse, τ is independent of the environment as are output actions, but is classified as input, since it can be an initial action in a prisum! Since [CW95] deals with process priority rather than action priority, comparison with PCBS cannot be exact. But it does appear that the complexity of [CW95] comes from the handshake model.

15   Examples

Let I be a set of numbers. Then $\prod_{i \in I} i!_i \, \mathbf{0}$ sorts I. The number of priority levels this program needs depends on I. Hardware implementations typically provide only a limited number of priority levels, so the programs that follow use only a bounded number of them, independent of the input. Correctness is not dealt with in these examples; their purpose is to explore the power of the language.

Example 13 The guarded sum is an interrupt operator. The motivating example from [CH90] and [CW95] is a counter that accepts "Up" and "Down" commands until interrupted. Here is a variation, a clock that counts off intervals until interrupted by any signal at priority 0 or 1.

$$clock \ n \ = \ x? \, \mathbf{0} \, \& \, (n!_1 \, (clock \ \ (n + 1)))$$

To be sure to stop the clock, the signal from outside must be sent at priority 0.

Example 14 Mixing functional and concurrent programming.

$$add\ n\quad = n!_0\ v?\ (add\ \ (n+v))$$
$$fib\qquad = test\ \ (0!_0\ (add\ 1\,))\ (seq\ fib)\ \ lefttrees$$
$$seq\ [\,]\quad = \mathbf{0}$$
$$seq\ \ (l{:}ls) = l!_1\ (seq\ ls)$$

fib is a Kahn network. The output is fed back at priority 1, and so has to synchronise with the adder's pauses. *fib* is sure to work only with *lefttrees*, which always prefers the left process. Other oracles might choose the output when it is not ready; the program will then loop.

Example 15 Priorities simplify data types and save on transmissions.

$$gen\ n\ \ = n!_1\ (gen\ (n+1\,))$$
$$trap\qquad = n?\ n!_0\ (\phi_n\ trap)$$
$$\pi_{\phi_n}\qquad = 0$$
$$\phi_n\qquad = \{x\uparrow x,\ x\downarrow\text{if } n\ divides\ x\ \text{then } \tau\ \text{else } x\}$$
$$primes = test\ trap\ \ (gen\ 2)\ lefttrees$$

The prime numbers are listed as echoes from *trap*. For each prime, it wears a further translation layer to become deaf to all multiples of this number. Without priorities, *gen* would have to wait to hear from *trap*, which must make an announcement for every number, so a type *Composite | Prime Int* would be needed.

Example 16 Priorities can order transmissions. This program lists primes in the range $[2 \mathinner{\ldotp\ldotp} l]$, by systematically testing each i for divisibility by all numbers less than itself. Without priorities, the primes may not come out in order.

$$gen\ i\ l\qquad = i!_1\ \text{if } i=l\ \text{then } \mathbf{0}\ \text{else } gen\ \ (i+1\,)\ \ l$$
$$cell\ n\qquad = i?\ \text{if } i=n\ \text{then } n!_0\ \mathbf{0}$$
$$\qquad\qquad\qquad \text{else if } i\ divides\ n\ \text{then } \mathbf{0}\ \text{else } cell\ n$$
$$primes\ l\ ots = test\ \ (pars\ \ (map\ cell\ [2 \mathinner{\ldotp\ldotp} l]))\ \ (gen\ 2\ l)\ ots$$

Example 17 Broadcast sort revisited. Recall that in the unprioritised version, the input was assumed to have no duplicates. The input list [1,2,2,3] would be sorted correctly into $in\ (-,1)\,|\,in\ (1,2)\,|\,in\ (2,2)\,|\,in\ (2,3)\,|\,in\ (\top,3)$, but things go wrong in the output phase. The output could come out as [1,2,3,2] since both $out\ (2,2)$ and $out\ (2,3)$ are triggered by the 2 from $out\ (1,2)$. A prioritised version overcomes this: $out\ (l,u)$ announces u when it hears l, at high priority if $l=n$, and at low priority otherwise. This ensures that $out\ (2,2)$ speaks before $out\ (2,3)$. The changes needed to the unprioritised version are small.

Example 18 Distributed backtrack: The eight queens problem. Recall the problem: place eight queens on a chessboard so that no two hold each other in check; that is, no two may lie on the same row, column, or diagonal.

The actions of the program are given by

$$act ::= Place\ sq\ \Big|\ Revoke\ i$$

where sq ranges over the squares of the board, and i over integers. Let n be the size of the board.

$$
\begin{array}{ll}
\textit{free sq i n} & = x?\,\text{if } \textit{check sq x} \text{ then } \textit{checked sq i n} \\
& \qquad\quad \text{else } \textit{free sq i n} \\
& \qquad\quad \&\, (\textit{Place sq})\,!_{n-i}\,(\textit{placed sq i n}) \\
\textit{checked sq i n} & = (\textit{Revoke i})\,?\,(\textit{free sq i n}) \\
\textit{placed sq 0 n} & = (\textit{Revoke 0})\,!_n\,\mathbf{0} \\
\textit{placed sq i n} & = (\textit{Revoke i})\,!_{n-1}\,(\textit{checked sq }(i-1)\ n) \\
\textit{queens n} & = \prod_{sq}(\textit{free sq 0 n})
\end{array}
$$

Each square is a process, *free* to start with. Free squares try to grab a piece; one succeeds. As a result, others may become *checked* and will then await a *Revoke* by the square that checked them. The program does a depth first search. The deeper the search, the higher the priority of the processes. Eventually, no free square remains, perhaps before the maximum number of pieces have been placed. Then the last placed piece succeeds in doing a low priority *Revoke*. A revoked square pretends it was checked by the previous piece, thus avoiding looping. It will not be eligible for placement until the configuration preceding it has changed. The square that placed itself first retires altogether when it revokes, because all combinations with a queen there have been tried.

The program puts out sequences of *Place*s and *Revoke*s till all squares have retired. The first solution is often found quickly, with few or no *Revoke*s. Finding all solutions takes the same total time no matter how the non-determinism is resolved.

archive below a printer process. It prints all maximal solutions, and all locally maximal ones prior to the first. It prints at priority 0 to avoid loss of information from *queens*, which has maximum priority 1.

$$
\begin{array}{ll}
\textit{archive l n} & = x?\,\text{case} \qquad x \text{ of} \\
& \qquad\quad \textit{Place sq} \;\rightarrow\;\; \textit{archive }(sq\!:\!l)\ n \\
& \qquad\quad \textit{Revoke i} \;\rightarrow\;\; \text{if } \big|\,l\,\big| \geq n \\
& \qquad\qquad\qquad\qquad\quad \text{then } l!_0\,\big(\textit{archive }(\textit{tail l})\,\big|\,l\,\big|\big) \\
& \qquad\qquad\qquad\qquad\quad \text{else } \textit{archive }(\textit{tail l})\ n \\
\textit{solution n ots} & = \textit{test }(\textit{archive }[\ ]\ 0)\ (\textit{queens n})\ \textit{ots}
\end{array}
$$

Deprioritisation can be used to ensure that *archive* has highest priority, no matter what the algorithm in *queens*.

A variant of *queens* [Pra94] uses only two priority levels.

16 Conclusions and Future Work

From early work. The essential features of unbuffered broadcast communication have been captured in a simple, natural and well behaved process calculus. Along the way, light has been shed on concepts such as autonomy, audibility and runs of a process. The mathematical apparatus of operational semantics carries over, with reinterpretation, but theories of observation need considerable reworking.

38

From CBS$^+$. A single channel [Pra93a] makes for elegant notation. It makes the pure calculus a special case of pattern matching on values transmitted. It also makes it easy to see restriction and hiding as special cases of translation, and leads to the notion of context reversal. The + operator gives CBS$^+$ the advantage of a familiar algebraic framework.

It is misleading that weak bisimulation for CBS has exactly the standard form; it took a long time to define it to produce the laws arrived at by a notion of testing (see [Pra91, Pra93a]). CBS$^+$ has an additional clause matching losses with hearing, yielding weak laws without τ's.

From current CBS. The implementation of CBS$^+$ filters out uninteresting values heard by a process. Moving this task to the user, and restricting choice to the form $x?\,p_1 + w!\,p_2$ (the only form needed in practice with CBS$^+$), led to CBS [Pra93b], a simpler calculus (two kinds of action instead of three) where in addition $\approx$ is a congruence.

Programming with broadcasts. This new paradigm expresses concisely much that would be tedious in CCS (or possibly inexpressible, see [Hol93, Jen94]), and yields interesting new algorithms.

Simulators for CBS are simple; a key fact in ongoing work to prove executable CBS programs. CBS can be used to annotate parallel evaluation in the host language; meaningful experiments with a parallelism profiler are possible. CBS is simply typed; translators and existential types make an abstraction facility.

Priorities. These are added easily [Pra94] to CBS, yet extend the power of the calculus significantly. As in CBS, some of the conciseness is due to an encoding of negative premises into hearing, but the main reason for the simplicity of PCBS is that priorities attach meaningfully only to autonomous actions.

Process calculi. The overwhelming predominance of handshake communication may be a historical accident. The model offers no obvious way to run processes. It appears incapable of distributed implementation; see for example [Sjö91]. Prioritised CBS compares strikingly with attempts to add priorities to CCS. There is increasing willingness to look at other models, and CBS offers one.

Ongoing and future work. There are many lacunae in this paper. A complete axiomatisation of $\approx$, a characterisation in terms of testing, an abstract machine for CBS, an operational study of contexts, and a proof system to deal with both CBS and the host language are obvious. Work is progressing on all these, and has in fact gone quite far in some cases. A timed CBS is being developed, and a higher order calculus is in its early stages. Small applications are being planned to evaluate the usefulness of CBS.

Acknowledgements. I thank my colleagues Ed Harcourt, Pawel Paczkowski and Martin Weichert, and an anonymous referee for detailed comments on this paper. CBS has now been in development for more than 6 years, and owes something to almost everyone I know in the fields of concurrency and process calculi.

References

[Abr70]   Norman Abramson. The Aloha system—another alternative for computer communications. In *FJCC*, pages 281–285, 1970.

[Abr87]   Samson Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53, 1987.

[BC91]   Kenneth Birman and Robert Cooper. The ISIS project: Real experience with a fault tolerant programming system. *Operating Systems Review*, 25(2), April 1991.

[BG92]   Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19, 1992.

[BKT84]   J. A. Bergstra, J. W. Klop, and J. V. Tucker. Process algebra with asynchronous communication mechanisms. In *Seminar on Concurrency*, pages 76–95. Carnegie-Mellon University, July 1984. Springer Verlag LNCS 197.

[BMT92]   Dave Berry, Robin Milner, and David Turner. A semantics for ML concurrency primitives. In *Symposium on Principles of Programming Languages*. ACM, 1992.

[Bro88]   Manfred Broy. Broadcasting buffering communication. *Comput. Lang.*, 13(1):31–47, 1988.

[BS94]   Bernard Berthomieu and Thierry Le Sergent. Programming with behaviours in an ML framework; the syntax and semantics of LCS. In *ESOP*, April 1994. Springer Verlag LNCS 788.

[Bur88]   F. W. Burton. Nondeterminism with referential transparency in functional languages. *The Computer Journal*, 31(3):243–247, 1988.

[BW90]   J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 1990.

[CG89]   Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[CH90]   Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. *Information and Computation*, 87, 1990.

[CNL89]   S. T. Chanson, G. W. Neufeld, and L. Liang. A bibliography on multicast and group communication. *Operating Systems Review*, 23(4), October 1989.

[CW95]   Juanito Camilleri and Glynn Winskel. CCS with priority choice. *Information and Computation*, 116, 1995.

[DK86]   Rina Dechter and Leonard Kleinrock. Broadcast communications and distributed algorithms. *IEEE Trans. on Computers*, 35(3):418, Mar 1986.

[dNH84]   Rocco de Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83, 1984.

[Geh84]   Narain Gehani. Broadcasting sequential processes. *IEEE Trans. on Software Engg.*, 10(4):343, July 1984.

[Gim95]   Eduardo Giménez. Implementation of co-inductive types in Coq: An experiment with the alternating bit protocol. Internal report, ENS Lyon, June 1995.

[GMP89]   Alessandro Giacalone, Prateek Mishra, and Sanjiva Prasad. Facile: A symmetric integration of concurrent and functional programming. *International Journal of Parallel Programming*, 18(2), 1989.

[Gro90]   J.F. Groote. Transition system specifications with negative premises. In *CONCUR '90*, 1990. Springer Verlag LNCS 458.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[Hen90] Matthew Hennessy. CSP with value-passing. Technical Report HPL-ISC-TM-90-025, Hewlett Packard Ltd., 1990.

[HL93] Matthew Hennessy and Huimin Lin. Proof systems for message-passing process algebras. In *CONCUR*, 1993. Springer Verlag LNCS 715.

[HL95] Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138, 1995.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Hol83] Sören Holmström. PFL: A functional language for parallel programming. Technical Report 7, Dept. of Computer Sciences, Chalmers Univ. of Tech., 1983.

[Hol93] Uno Holmer. Translating broadcast communication into SCCS. In *CONCUR*, August 1993. Springer Verlag LNCS 715.

[HR95] Matthew Hennessy and Julian Rathke. Bisimulations for a calculus of broadcasting systems. In *CONCUR*, 1995. Springer Verlag LNCS.

[HT92] Tzung-Pei Hong and Shian-Shyong Tseng. Parallel perceptron learning on a single-channel broadcast communication model. *Parallel Computing*, 18:133–148, 1992.

[Jen94] Clause Torp Jensen. Interpreting broadcast communications in CCS with priority choice. In *Proceedings of 6th Nordic Workshop on Programming Theory*, BRICS Report Series, Aarhus University, October 1994.

[JJH90] He Jifeng, Mark Josephs, and C.A.R. Hoare. A theory of synchrony and asynchrony. Technical report, Programming Research Group, Oxford University Computing Laboratory, January 1990.

[Jon93] Simon Jones. Translating CBS to LML. Technical report, Department of Computer Science, Chalmers University of Technology, 1993.

[JU90] Mark Josephs and Jan Udding. Delay-insensitive circuits: an algebraic approach to their design. In *CONCUR '90*, 1990. Springer Verlag LNCS 458.

[KH94] Josva Kleist and Martin Hansen. Process calculi with asynchronous communication. Technical report, Dept. of Computer Science, Aalborg Univ., August 1994.

[Kor94] Henri Korver. A theory for simulators. *Computer Journal*, 37(4), 1994.

[Lar87] Kim Larsen. A context-dependent equivalence between processes. *Theoretical Computer Science*, 49, 1987.

[LT87] Nancy Lynch and Mark Tuttle. Hierarchical correctness proofs for distributed algorithms. Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, Massachusetts Institute of Technology, 1987.

[MB76] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7), July 1976.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MP85] J. C. Mitchell and G. D. Plotkin. Abstract types have existential types. In *ACM Conference on Principles of Programming Languages*, pages 37–51, 1985.

[NCCC94] V. Natarajan, I. Christoff, L. Christoff, and R. Cleaveland. Priority and abstraction in process algebra. In *FST&TCS*, 1994. Springer Verlag LNCS 880.

[Pet94]   Jenny Petersson. Tools for a calculus of broadcasting systems. Licentiate thesis, Department of Computer Science, Chalmers University of Technology, 1994.

[Pnu85]   Amir Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Springer Verlag LNCS 194*. ICALP, 1985.

[Pra87]   K. V. S. Prasad. *Combinators and Bisimulation Proofs for Restartable Systems*. PhD thesis, University of Edinburgh, December 1987.

[Pra89]   K. V. S. Prasad. On the non-derivability of operators in CCS. Technical Report 55, Dept. of Computer Sciences, Chalmers Univ. of Tech., Dec 1989.

[Pra91]   K. V. S. Prasad. A calculus of broadcasting systems. In *TAPSOFT, Volume 1: CAAP*, April 1991. Springer Verlag LNCS 493.

[Pra93a]   K. V. S. Prasad. A calculus of value broadcasts. In *PARLE*, June 1993. Springer Verlag LNCS 694.

[Pra93b]   K. V. S. Prasad. Programming with broadcasts. In *CONCUR*, August 1993. Springer Verlag LNCS 715.

[Pra94]   K. V. S. Prasad. Broadcasting with priority. In *ESOP*, April 1994. Springer Verlag LNCS 788.

[Pra95]   K. V. S. Prasad. Broadcasting in time. Technical report, Department of Computer Science, Chalmers University of Technology, June 1995. Preliminary version.

[Rep91]   J. H. Reppy. A higher order concurrent language. *SIGPLAN Notices*, 26(6):294–305, 1991. ACM SIGPLAN'91 Conference on Programming Language Design and Implementation.

[RW93]   C. Runciman and D. Wakeling. Profiling parallelism. Internal report, Department of Computer Science, University of York, 1993.

[Seg92]   Roberto Segala. A process algebraic view of input/output automata. Technical Report MIT/LCS/TR-557, MIT, October 1992.

[Sjö91]   Peter Sjödin. *From LOTOS specifications to distributed implementations*. PhD thesis, Uppsala University, December 1991.

[SK88]   M. Sloman and J. Kramer. *Distributed Systems and Computer Networks*. Prentice Hall, 1988.

[SNP87]   R. K. Shyamasundar, K. T. Narayana, and T. Pitassi. Semantics for nondeterministic asynchronous broadcast networks. Technical report, Pennsylvania State Univ., March 1987.

[YLC90]   Chang-Biau Yang, R. C. T. Lee, and Wen-Tsuen Chen. Parallel graph algorithms based upon broadcast communications. *IEEE Trans. on Computers*, 39(12):1468, Dec 1990.

This article was processed using the LaTeX macro package with LLNCS style