A Calculus of Broadcasting Systems

K.V.S.Prasad
Department of Computer Sciences
Chalmers University of Technology
S-412 96 Gothenburg, Sweden
email: prasad@cs.chalmers.se

Abstract

Local area networks (LANs) and everyday speech inspire a model of communication by unbuffered broadcast. Computation proceeds by a sequence of messages, each transmitted by one agent and received by zero or more others. Transmission is autonomous, but reception is not. Each message is received instantaneously by all agents except the transmitter, but is read only by those who were monitoring it at the time; others discard it. As in CCS, agents learn about the environment only through the messages they read. Programming such a system is hard because we have to ensure that messages are read.

Testing resembles a viva-voce examination. Observation, restriction and hidden actions differ from their CCS counterparts, as does testing equivalence.

We capture this model in a Calculus of Broadcasting Systems (CBS). We use transition systems with transmit, read and discard actions. Discards are self-loops, and only auxiliary. We have some results on strong bisimulation and testing, but much work remains to make CBS tractable.

1 Preliminaries

The setting The natural means of communication in local area networks (LANs) has always been *broadcast* [Abr70] [MB76], not point-to-point message passing. But theories like CSP[Hoa85] and CCS[Hen88][Mil89] deal only with the latter, even though they developed at the same time as LANs[Hoa78][Mil80]. Most books on distributed systems, too, [SK88] for example, treat broadcast only as a hardware feature, not as a programming primitive. This mismatch would appear firstly to throw away a lot of the communication bandwidth. That it might also throw away simple solutions to some programming problems is suggested by the several algorithms that use broadcast, and the great interest [CNL89] in implementations of (reliable) broadcasting.

Previous semantic studies of broadcast programming include an informal study [Geh84], and denotational semantics for broadcasting [SNP87] [Bro88]. ESTEREL[BCG86] is an imperative, synchronous, deterministic language with broadcast as a programming primitive. LINDA (Chap. 8 of [BA90] gives a description) is in effect a programming model using broadcast. ESTEREL has been given a behavioural semantics. I know of no theory of broadcast communication giving equivalences, etc., in the process calculus tradition.

Buffered and unbuffered broadcast Both [Bro88] and LINDA are *buffered broadcast* systems, like mass mailing: receivers can pick up messages any time after they have been broadcast.

Unbuffered broadcast, the topic of this paper, is like radio or TV communication, or everyday speech in some situations. Each *message* has one *transmitter*, and zero or more *receivers*. Messages are received instantaneously by all receivers (i.e., by the antenna). The receiver can choose whether to read, but it is the transmitter that determines what is received and when.

Broadcast in process calculus Broadcast cannot be described unless actions are divided into transmissions and receptions. So the following "broadcast" operator due to Harel[Pnu85] would be better called "enforced multiway synchronisation". Note the symmetric role played by all participants.

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

$$\frac{E \xrightarrow{a} E' \quad F \overset{a}{\not\longrightarrow}}{E \mid F \xrightarrow{a} E' \mid F} \qquad \frac{E \xrightarrow{a} E' \quad F \overset{a}{\not\longrightarrow}}{F \mid E \xrightarrow{a} F \mid E'}$$

Harel's operator does capture two aspects of broadcast: the multiway synchronisation, and the fact that it is unreasonable for someone listening for a message to discard it when it appears. The negative premise ensures this does not happen.

Overview of this paper Section 2 develops an informal model (PA) of broadcasting. Section 3 formalises PA into a calculus (CBS), and checks that the transition system has the properties required. Section 4 gives basic bisimulation results. Section 5 is a first look at testing and observational equivalence in CBS. It gives an example of the use of tests as specifications. Section 6 discusses some design issues, and contrasts CBS and CCS.

2 Informal models of broadcasting

Concepts from CCS We shall see systems as consisting of *agents* whose identity persists through time, and whose *behaviour* consists of discrete communication *actions*, *transmission* or *reception* of messages. Agents evolve from state to state via these actions. There is no formal distinction between *agent* and *state*. Agents can be built up out of other agents, so we do not distinguish between agents and systems. They learn about their environment only through communication.

2.1 Local area networks (LANs)

Messages consist of headers and contents (*values*). The header (*name* or *signal*) specifies who is to *read* the message. Each message is received by all systems connected to the network. Each receiver *monitors* certain headers. It examines each message and reads it if the header is one it is monitoring, otherwise it *discards* it. This is sometimes called *multicast*. A message may be read by any subset of the agents in a system. Usually, only one agent can broadcast at a time. This is called the *single channel assumption*.

If two agents try to broadcast simultaneously over a LAN, a collision occurs, and no message is transmitted. There are well known algorithms to deal with collisions, typically using random delays before attempting to transmit again. We will not model this level of detail, but a level above it that already incorporates collision detection and resolution.

A straightforward formalisation of LANs in a CCS like framework leads to CBS. The model below is more intuitive, and ensures that we are free of incidental details of existing LAN architectures.

2.2 Speech as broadcast communication

Speech is very like unbuffered broadcast. It is *autonomous*; no one has to listen. Hearing is *controlled*; it happens only when someone speaks. Everybody within earshot *hears* everything that is said, but only those who wish to *listen* do so; others *ignore* what they hear. Even collisions, detection and resolution are features of conversation. Hearing is a *silent* activity, and cannot be directly observed by others. If the hearing was listening, it might be deduced from later utterances by the hearer.

Everyday speech does not have a single channel, address groups, and anonymous participants. These features can be found in public address (PA) systems.

The public address (PA) model Imagine a PA system in a friendly airport where anyone can hand in a message. Messages can only be read out one by one, so there may be a priority system to choose among competing messages. Everyone monitors certain signals. Mr. K, catching BA412, listens for "BA412" and "Mr. K", as well as "bomb", say. He listens to messages with these names in them; the rest he hears and ignores.

Most signals specify several listeners: messages about BA412 are for all those travelling by it. "Would Mr. K please meet Ms. O at the desk" indicates both source and intended listener.

3 Formalising the model

We now design CBS to mimic the behaviour of PA. We represent agents and changes of state by labelled transition systems and communicating agents as being in parallel with each other. CBS agents are very like those in CCS as far as dynamic operators go. The static operators differ because we use broadcast instead of handshaking. CBS describes automata communicating by "speech".

We shall motivate the various rules, referring forward to the summary in Table 4. We write $a\,(v)$ for the *message* with *name* or *signal* a and *value* v.

3.1 Actions and action prefixes

The rules for action prefix are given in the first three rows of Table 4. Ignore the τ rules for now.

$a\,(v)\,!$ is the *transmit* prefix. The agent $a\,(v)\,!\,P$ can do only one *action*, a *transmission*. It transmits $a\,(v)$ and becomes P. We write $a\,(v)\,!\,E \xrightarrow{a(v)!} E$.

$a\,(x)\,?$ is the *monitor* prefix for the name a. The agent $a\,(x)\,?\,Q\,(x)$ will *read* any message it receives with this name, and become $Q\,(v)$. We write $a\,(x)\,?\,Q\,(x) \xrightarrow{a(v)?} Q\,(v)$.

0 has nothing to transmit or monitor.

3.2 Discards

$a!\,P$ will only transmit, and *discards* all messages it receives. $a\,(x)\,?\,Q\,(x)$ monitors only a, and discards messages with any other name. **0** discards everything it receives. These rules are all in the discard column of Table 4.

For any agent in PA, the set of signals it monitors is well defined. So its response to any message is also well defined: read it if the signal was monitored, discard it otherwise. Thus we expect $a\,(v)\,?$ and $a\,(v)\,:$ to be mutually exclusive and exhaustive. Since agents do not change state as a result of discards, we could build CBS using the predicate "not monitoring a"; essentially the same calculus results. Discards reflect reality at a lower level than the predicate.

In state diagrams, discards are self-loops. The : arcs can be deduced from the ?'s. If a state does not have an $a?$ leading out from it, it has an $a:$ self-loop. These discard self-loops are therefore never shown, only ? and ! actions are. We will not have $a:P$ in the syntax.

3.3 Example: a cat, her owner, and his friend

$MEIOSIS \stackrel{\text{def}}{=} meiosis?\, miao!\, MEIOSIS$

$OWNER \stackrel{\text{def}}{=} meiosis!\, miao?\, ha?\, SUCC$

$FRIEND \stackrel{\text{def}}{=} meiosis?\, miao?\, ha!\, \mathbf{0}$

$CATSYSTEM \stackrel{\text{def}}{=} MEIOSIS \mid OWNER \mid FRIEND$

Table 1: Cat, owner and friend

In Table 1, Meiosis is a cat who answers to her name. Her owner can prove this by doing the test *OWNER*, where *SUCC* is a state denoting that the test has succeeded. The *ha* comes from the amused friend.

Thus at the start the address group monitoring *meiosis* is *MEIOSIS* and *FRIEND*, while the address group *miao* is empty (at this stage none of our three protagonists is listening for this signal). We will define $\mid$ to allow

$$MEIOSIS \mid OWNER \xrightarrow{meiosis!} miao!\, MEIOSIS \mid miao?\, ha?\, SUCC$$

$$CATSYSTEM \xrightarrow{meiosis!} miao!\, MEIOSIS \mid miao?\, ha?\, SUCC \mid miao?\, ha!\, \mathbf{0}$$

Note that unary operators bind tighter than binary ones.

3.4 Parallel composition

Broadcast We derive the second transition above, showing that both cat and friend hear the owner's *meiosis*, by using the communication rule in Table 2 twice. We have a binary parallel composition, rather than one of arbitrary arity, that simulates broadcast. We need the rule Join-read to make $\mid$ associative.

3.4.1 Discards and the parallel combinator

In [Win84], the interleaving actions of CCS are expressed as synchronous actions by both components, but with one idling. This is exactly what we do in the interleaving rule, except that instead of a general idling we have a selective discard of the message named.

A message can be read by one receiver and discarded by another (one application each of the communication and interleave rules). The read-discard rule ensures associativity here.

Enforced reading The $\xrightarrow{a(v):}$ premise in the interleaving rule ensures that exactly one of the two rules, interleave or communicate, will apply, depending on whether the

Communication	$\dfrac{E \xrightarrow{a(v)!} E' \quad F \xrightarrow{a(v)?} F'}{E \mid F \xrightarrow{a(v)!} E' \mid F'}$	and commutative pair
Join-read	$\dfrac{E \xrightarrow{a(v)?} E' \quad F \xrightarrow{a(v)?} F'}{E \mid F \xrightarrow{a(v)?} E' \mid F'}$	
Interleave	$\dfrac{E \xrightarrow{a(v)!} E' \quad F \xrightarrow{a(v):} F}{E \mid F \xrightarrow{a(v)!} E' \mid F}$	and commutative pair
Read-discard	$\dfrac{E \xrightarrow{a(v)?} E' \quad F \xrightarrow{a(v):} F}{E \mid F \xrightarrow{a(v)?} E' \mid F}$	and commutative pair
Join-discard	$\dfrac{E \xrightarrow{a(v):} E \quad F \xrightarrow{a(v):} F}{E \mid F \xrightarrow{a(v):} E \mid F}$	

Table 2: Rules for parallel composition

receiver is monitoring the message. This premise and the negative premise $\xrightarrow{a(v)?}\!\!\!\!\!\!/$ are both equivalent to "not monitoring a".

With negative premises, we would expect to be able to derive them first, independently of the positive transitions. (For an explanation of this "stratification" technique and other details see [Gro90]). So we expect to be able to derive discards independently of other transitions.

Synchronisation algebra The eight rules in Table 2 can be compressed into the single rule of Table 4, using a *synchronisation algebra*[Win84] of actions. This is the advantage of discards over negative premises.

Since there can only be one message at a time, we do not define multiplication between actions unless they refer to the same message. The product of two transmissions is always undefined. This ensures that every transmission can be registered.

$\circ$ is commutative and associative. The latter is easy to check because there is a ranking: $a(v)!$ first, then $a(v)?$ and $a(v):$ last. To multiply two actions is to pick the higher ranking of the two. [Win84] shows that if the synchronisation algebra is commutative and associative, so is $\mid$. We went the other way.

3.5 Sum and recursion

Recursion and sum are as in CCS for transmits and reads, but not for discards.

$$E \stackrel{\text{def}}{=} CH\backslash\{c\} \mid F[\phi]$$
$$F \stackrel{\text{def}}{=} f!\,F$$
$$CH \stackrel{\text{def}}{=} c!\,C \mid h?\,H$$

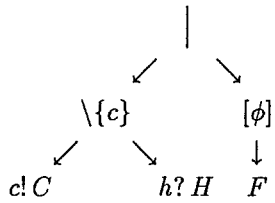

Table 3: A small tower of Babel

$a?\,P + b!\,Q$ will say b unless preempted by an a from someone else. If it receives a c, both branches will discard the message. The usual sum rule here would drop one branch, thus changing state on a discard. Hence the new rule sum-discard. If only one branch can discard a message, the other can read it, so the familiar + rule applies.

The recursion rule for discards is a special case of the usual one. It lets us derive $\text{rec}X.\,b?\,X \xrightarrow{a:} \text{rec}X.\,b?\,X$ from $b?\,X \xrightarrow{a:} b?\,X$. The usual recursion rule would let us derive $\text{rec}X.\,b?\,X \xrightarrow{a:} b?\,X$, which is irritating, since we would like to capture syntactically the fact that discards never cause a change of state.

We still have a problem with $\text{rec}X.\,X$, which can do neither $a?$ nor $a:$. To ensure that discards and reads are duals, recursion for discards should take the largest fixed point instead of the least. If we restrict ourselves to guarded recursion, we can get away with the simpler rule in Table 4.

3.6 Renaming and Scoping

The static structure of systems Scoping and renaming are static unary operators as in CCS. It would be reasonable to restrict a subsystem to only listening (or only transmitting) on some names, but we simply scope transmission and reading together. We take renaming to be a bijection.

Table 3 shows a tree structured PA system in a tower of Babel. All nodes speak English except F, which speaks French. c and h are the topics of cricket and history, and ϕ a translation from French to English. $\phi(f) = h$. The leaves are agents built from dynamic operators. All other nodes are labelled by scoping or renaming, and under each such are all the subsystems governed by it, in parallel. The root could be labelled by just $\mid$. Note that prefixes bind tighter than renaming or scoping.

The subsystem $CH\backslash\{c\}$ does not expect anyone else to be interested in cricket, but h is of interest to all subagents of E. So the $c!$ request is sent only up to $\backslash\{c\}$, which broadcasts $c!$ to all its subnodes. The $\phi(f!) = h!$ request goes upto E, getting translated as it passes up through the renaming node. Messages broadcast down through such a node are simultaneously interpreted.

Rules A $c!$ by $CH\backslash\{c\}$ is *hidden* outside the node. Being autonomous, transmission cannot be restricted away. Instead, it is translated into a silent action, $\tau!$, which no agent can read. Thus the rules hide-transmit and expr-discard in Table 4. As with internal actions in CCS, we need only one τ. The notation $\tau!$ emphasises that this is a transmission, so autonomous, but hidden. It is not quite the CCS τ.

The hide-discard rule says that $CH\backslash\{c\}$ ignores any c it receives.

In other cases, the usual CCS restrict rule applies. Thus $P\backslash\{a\}$ has all $a?$-paths from the state P pruned (replaced by a: self-loops), and all $a!$-arcs renamed to $\tau!$.

3.7 Syntax and semantics of CBS

Let $\mathcal{S}$ be the set of signals or message names, and $\tau \notin \mathcal{S}$ be a special name. Let V be the set of values passed between agents. Let $\mathcal{S}_V = \mathcal{S} \times V$. Then the set of messages is $\mathcal{S}_V \cup \{\tau\}$, and $Act = (\mathcal{S}_V \times \{!, ?\}) \cup \{\tau!\}$ is the set of transmissions and reads. The elements of Act are called *real actions*. The arcs of synchronisation trees are labelled by real actions. If there is no danger of confusion, we say "actions" to mean real actions. The set of all actions, both real ones and discards, is $Act^\cdot = Act \cup (\mathcal{S}_V \times \{:\}) \cup \{\tau:\}$.

Let $a, b, c, \ldots$ range over $\mathcal{S}$; u, v over V; μ over Act; and ν over $Act^\cdot$.

The function $sign: Act^\cdot \to \{!, ?, :\}$ says whether an action is a transmission, a read, or a discard. The function $message: Act^\cdot \to \mathcal{S}_V \cup \{\tau\}$ throws away the sign of an action and returns the message. The function $name: \mathcal{S}_V \cup \{\tau\} \to \mathcal{S} \cup \{\tau\}$ throws away the value carried by a message. We abuse notation and apply *name* also to actions, writing $name(\nu)$ to mean $name(message(\nu))$.

The syntax of CBS expressions is as follows. Recursion must be guarded.

$$E ::= X \mid \mathbf{0} \mid \tau!\,E \mid a(v)!\,E \mid a(x)?\,E \mid E + E \mid E \mid E \mid E\backslash\mathcal{N} \mid E[\phi] \mid \mathrm{rec}X.\,E$$

Here E is an expression, X a variable, $\mathcal{N} \subseteq \mathcal{S}$ a set of message names, and x a variable internal to an agent. We represent by $\mathbf{E}$ the set of expressions, and by $\mathbf{P}$ the set of *agents*, i.e., the closed expressions of CBS. Note that $a:P$ is not part of the syntax, nor is $\tau?\,P$.

Renamings $\phi: \mathcal{S} \cup \{\tau\} \to \mathcal{S} \cup \{\tau\}$ are bijections over $\mathcal{S}$ and $\phi(\tau) = \tau$. Renaming extends naturally to actions. It cannot change the sign of an action.

Since we do not have infinite sums, we cannot encode value passing in the pure calculus unless V is finite.

The operational rules for CBS are given together with the synchronisation algebra of actions in Table 4. Note that some of the rules are applicable in more than one of the columns; this is indicated by broken vertical rules between the columns. The parallel, restrict, and rename rules apply to all three kinds of action; the first two sum rules and the first recursion rule to both transmission and read.

Syntax: $E ::= X \mid \mathbf{0} \mid \tau!E \mid a(v)!E \mid a(x)?E \mid E+E \mid E \mid E \mid E\backslash\mathcal{N} \mid E[\phi] \mid \mathrm{rec}X.E$

signals $a, b \in \mathcal{S}$ values $u, v \in V$ Recursion must be guarded

$\mathcal{S}_V = \mathcal{S} \times V \quad \mu \in Act = (\mathcal{S}_V \times \{!,?\}) \cup \{\tau!\} \quad \nu, \nu_1, \nu_2 \in Act' = (\mathcal{S}_V \times \{!,?,:\}) \cup \{\tau!, \tau:\}$

Operator	Transmit	Read	Discard
0			$\mathbf{0} \xrightarrow{a(v):} \mathbf{0}$
Expr			$E \xrightarrow{\tau:} E$
Transmit	$\tau!E \xrightarrow{\tau!} E$ $a(v)!E \xrightarrow{a(v)!} E$		$\tau!E \xrightarrow{a(v):} \tau!E$ $a(v)!E \xrightarrow{b(u):} a(v)!E$
Monitor		$a(x)?E(x) \xrightarrow{a(v)?} E(v)$	$\dfrac{}{a(x)?E \xrightarrow{b(v):} a(x)?E}\ a \neq b$
Sum	$\dfrac{E \xrightarrow{\mu} E'}{E+F \xrightarrow{\mu} E'}$	$\dfrac{E \xrightarrow{\mu} E'}{F+E \xrightarrow{\mu} E'}$	$\dfrac{E \xrightarrow{a(v):} E \quad F \xrightarrow{a(v):} F}{E+F \xrightarrow{a(v):} E+F}$
Parallel	$\begin{array}{c\|ccc} \circ & a(v)! & a(v)? & a(v): \\ \hline a(v)! & \text{undefined} & a(v)! & a(v)! \\ a(v)? & a(v)! & a(v)? & a(v)? \\ a(v): & a(v)! & a(v)? & a(v): \end{array}$	$\begin{array}{c\|ccc} \circ & \tau! & \tau: \\ \hline \tau! & \text{undefined} & \tau! \\ \tau: & \tau! & \tau: \end{array}$	$\nu_1 \circ \nu_2$ undefined if $message(\nu_1) \neq message(\nu_2)$
	$\dfrac{E \xrightarrow{\nu_1} E' \quad F \xrightarrow{\nu_2} F'}{E \mid F \xrightarrow{\nu_1 \circ \nu_2} E' \mid F'}\ \nu_1 \circ \nu_2$ defined		
Scope (Restrict)	$\dfrac{E \xrightarrow{\nu} E'}{E\backslash\mathcal{N} \xrightarrow{\nu} E'\backslash\mathcal{N}}\ name(\nu) \notin \mathcal{N}$		
(Hide)	$\dfrac{E \xrightarrow{a(v)!} E'}{E\backslash\mathcal{N} \xrightarrow{\tau!} E'\backslash\mathcal{N}}\ a \in \mathcal{N}$		$\dfrac{}{E\backslash\mathcal{N} \xrightarrow{a(v):} E\backslash\mathcal{N}}\ a \in \mathcal{N}$
Rename	$\dfrac{E \xrightarrow{\nu} E'}{E[\phi] \xrightarrow{\phi(\nu)} E'[\phi]}$		
Rec	Let Y stand for $\mathrm{rec}X.E$ and U for $E[\mathrm{rec}X.E/X]$ $\dfrac{U \xrightarrow{\mu} E'}{Y \xrightarrow{\mu} E'}$		$\dfrac{U \xrightarrow{a:} U}{Y \xrightarrow{a:} Y}$

Table 4: Syntax and semantics of CBS

3.8 Properties of the transition system

We show that discards can be derived first, without using any other transitions. This confirms their role as negative premises. Then we show that discards leave the state syntactically unchanged, and lastly that discards and reads are duals. $\equiv$ is syntactic equality.

Proposition 3.1 *Discards can be derived independently of other actions.*

Proof: Directly from the operational rules. Some rules for discards have no premises. The sum, restrict, rename and rec rules yield the same action in the conclusion as they use in the premise. For the parallel rule, $a(v)$: can result only if both components do $a(v)$:. ∎

Lemma 3.2 *If* $E \xrightarrow{a:} F$ *then* $E \equiv F$.

Proof: Directly from the discard rules. ∎

Lemma 3.3 *Discards and reads are mutually exclusive:* $E \xrightarrow{a(v):} E$ *iff* $E \xrightarrow{a(v)?} \not\rightarrow$, *and* $E \xrightarrow{\tau?} \not\rightarrow$. *(We already have* $E \xrightarrow{\tau:} E$.*)*

Proof: By induction on the structure of E.

τ first. If E is $\mathbf{0}$ or has an action prefix as the outermost operator $E \xrightarrow{\tau?} \not\rightarrow$ follows directly. Sum, restrict, rename and rec cannot produce a $\tau?$ unless a simpler expression does so. Hiding and parallel cannot produce a $\tau?$ at all.

Now for audible messages.

Base case: $\mathbf{0}$ discards all messages and reads none.

Step: Suppose $F \xrightarrow{a(v):} F$ iff $F \xrightarrow{a(v)} \not\rightarrow$ for all expressions F simpler than E.

If E has an action prefix outermost, the step is immediate.

If E is of the form $F + F'$, any message $a(v)$ it discards is discarded by both F and F', which cannot therefore read $a(v)$, by hypothesis. So $E + F$ cannot read $a(v)$. If E reads $a(v)$, one of F, F' must do so, therefore one of them cannot discard it, therefore E cannot discard it. Similar reasoning holds for E of the form $F \mid F'$; for E to read $a(v)$, at least one of F, F' must do so.

If E is $F\backslash\mathcal{N}$, we have two cases. If $a \notin \mathcal{N}$, E can read or discard messages exactly when F does. If $a \in \mathcal{N}$, then E can only discard messages of name a; it cannot read them.

Since renaming does not change the sign of actions, $F[\phi]$ satisfies the hypothesis if F does. Lastly, if E is $\mathrm{rec}X.F$, E can read or discard messages exactly when a simpler agent does. ∎

In the rest of this paper, we deal mostly with the pure calculus with no value passing.

4 Strong Bisimulation

Definition 4.1 *A binary relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a* strong bisimulation *if, whenever $P\mathcal{R}Q$ and $\nu \in Act^{:}$,*
(i) if $P \xrightarrow{\nu} P'$ then, for some Q', $Q \xrightarrow{\nu} Q'$ and $P'\mathcal{R}Q'$,
(ii) if $Q \xrightarrow{\nu} Q'$ then, for some P', $P \xrightarrow{\nu} P'$ and $P'\mathcal{R}Q'$ ∎

For any $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ we may define $\mathcal{F}(\mathcal{R})$ to be the set of pairs $\langle P, Q\rangle$ satisfying clauses (i) and (ii). Clearly, $\mathcal{R}$ is a strong bisimulation if $\mathcal{R} \subseteq \mathcal{F}(\mathcal{R})$.

Proposition 4.2 *There exists a largest strong bisimulation given by* $\sim = \cup\{\mathcal{R} \mid \mathcal{R} \subseteq \mathcal{F}(\mathcal{R})\}$. *$\sim$ is the maximum fixed point of $\mathcal{F}$, and is an equivalence relation.* ∎

If $P \sim Q$, we say P and Q are *strongly bisimilar.*

Proposition 4.3 *$P \sim Q$ iff for all $\nu \in Act^{:}$,*
(i) if $P \xrightarrow{\nu} P'$ then, for some Q', $Q \xrightarrow{\nu} Q'$ and $P' \sim Q'$,
(ii) if $Q \xrightarrow{\nu} Q'$ then, for some P', $P \xrightarrow{\nu} P'$ and $P' \sim Q'$ ∎

As long as we consider all actions including discards, the above standard theory is directly applicable. But we can reduce our work. The following lemma tells us that when proving strong bisimulation, we do not need to consider discards.

Lemma 4.4 *If $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a binary relation such that whenever $P\mathcal{R}Q$ and $\mu \in Act$,*
(i) if $P \xrightarrow{\mu} P'$ then, for some Q', $Q \xrightarrow{\mu} Q'$ and $P'\mathcal{R}Q'$,
(ii) if $Q \xrightarrow{\mu} Q'$ then, for some P', $P \xrightarrow{\mu} P'$ and $P'\mathcal{R}Q'$
then $\mathcal{R}$ is a strong bisimulation.

Proof: Suppose $P\mathcal{R}Q$ and $\mathcal{R}$ satisfies (i) and (ii). Now if $P \xrightarrow{a(v):} P$, we know by Lemma 3.3 that $P \xrightarrow{a(v)?} \not\rightarrow$. Then we know that $Q \xrightarrow{a(v)?} \not\rightarrow$. Otherwise P would have had to have been able to read $a(v)$, since reads are covered by the conditions on $\mathcal{R}$. By another application of Lemma 3.3, we have $Q \xrightarrow{a(v):} Q$. The target states after the matching discards $a(v):$ are already in the relation. Similarly for the other direction. Thus $\mathcal{R}$ is a strong bisimulation. ∎

Discards can be left out of state diagrams since they can be deduced from reads; now we know we can ignore all derivations where the last step is a discard. Discards are only needed in derivations when they contribute to a read or a transmit.

Proposition 4.5 *$\sim$ is a congruence for CBS, i.e., $P \sim Q$ implies $\mathcal{C}[P] \sim \mathcal{C}[Q]$ for any CBS context $\mathcal{C}$.*
Proof: By adapting the proof in [Mil89]. ∎

There are also general theorems from which we can conclude this proposition. The rules in CBS are both in the well-founded pure *tyft* format of [VG89], and in the GSOS format of [BIM88]. These papers prove that for calculi whose rules are in their respective formats, $\sim$ is a congruence. We can now quotient $\mathbf{P}$ by $\sim$.

Before we give a few bisimulation laws, we define sorts.

Definition 4.6 *For any $\mathcal{L} \subseteq \mathcal{S}$, if the real actions of P and all its derivatives have their names in $\mathcal{L} \cup \{\tau\}$ then we say P* has sort *$\mathcal{L}$, or $\mathcal{L}$ is a sort of P, and write $P{:}\mathcal{L}$.* ∎

This is a modification of the standard definition in two ways. The restriction to real actions has been motivated informally, and will soon be further justified. Note that if sorting were to deal with discards as well, every agent would have the sort $\mathcal{S}$.

We use names of actions rather than the actions themselves because we scope reads and transmits together. This produces a coarser sorting than in CCS, and makes the definition of the syntactic sort of an expression (by induction on its structure) slightly simpler than the standard one. We omit the details.

Proposition 4.7 Strong bisimulation laws

1. *(a)* $(\mathbf{P}/\sim, +, \mathbf{0})$ *is an abelian monoid.*
   (b) $P + P \sim P$
2. $\left(\mathbf{P}/\sim, \middle|, \mathbf{0}\right)$ *is an abelian monoid.*
3. *(a)* $P\backslash\mathcal{N}\backslash\mathcal{M} \sim P\backslash\mathcal{M}\backslash\mathcal{N} \sim P\backslash\mathcal{N} \cup \mathcal{M}$
   (b) $(a!\,P)\backslash\mathcal{N} \sim a!\,(P\backslash\mathcal{N})$ *if* $a \notin \mathcal{N}$, $\tau!\,P$ *otherwise.*
   $(a?\,P)\backslash\mathcal{N} \sim a?\,(P\backslash\mathcal{N})$ *if* $a \notin \mathcal{N}$, $\mathbf{0}$ *otherwise.*
   $(\tau!\,P)\backslash\mathcal{N} \sim \tau!\,(P\backslash\mathcal{N})$
   (c) $(P + Q)\backslash\mathcal{N} \sim P\backslash\mathcal{N} + Q\backslash\mathcal{N}$
   (d) $P\backslash\mathcal{N} \sim P$ *if* $\mathcal{L}(P) \cap \mathcal{N} = \emptyset$
   (e) $\left(P \middle| Q\right)\backslash\mathcal{N} \sim (P\backslash\mathcal{N}) \middle| (Q\backslash\mathcal{N})$ *if* $\mathcal{L}(P) \cap \mathcal{L}(Q) \cap \mathcal{N} = \emptyset$
4. *(a)* $(\mu P)\,[\phi] \sim \phi(\mu)\,P[\phi]$
   (b) $(P + Q)\,[\phi] \sim P[\phi] + Q[\phi]$
   (c) $\left(P \middle| Q\right)[\phi] \sim P[\phi] \middle| Q[\phi]$
   (d) $(P[\phi])\,[\psi] \sim P[\psi \circ \phi]$
5. $P[\phi]\backslash\mathcal{N} \sim (P\backslash[\phi]^{-1}\mathcal{N})\,[\phi]$

∎

5 Testing

Strong bisimulation is too strong to be an observational equivalence. For example, $a?\,\mathbf{0} \not\sim b?\,\mathbf{0}$, and yet we do not expect to tell such agents apart, since they say nothing. In this section, we take a first look at testing equivalence, using the ideas of [dNH84].

In the informal model, *tests* are just conversations between a tester and an agent, like viva-voce examinations held in public. The agent passes if the tester is satisfied by the conversation. $a?\,\mathbf{0}$ and $b?\,\mathbf{0}$ cannot make their presence felt. They can only satisfy testers who do not require the agent to speak.

Before we can define testing, we need to formalise conversation as computation. First we extend transitions to strings of actions. If s is the sequence $\nu_1, \nu_2, \ldots, \nu_n$, we write $P \xrightarrow{s} P'$ if there are agents $P_0 = P, P_1, P_2, \ldots, P_n = P'$ such that $P_i \xrightarrow{\nu_{i+1}} P_{i+1}$ for $0 \leq i \leq n$. We write $P \xrightarrow{s}$ to mean $\exists P'$ such that $P \xrightarrow{s} P'$.

5.1 Autonomy and computation

What makes transmission autonomous and reception controlled? Formally, nothing distinguishes the three kinds of transition so far, so *MEIOSIS* could do a *meiosis*? all by herself! The definition below rules this out. An isolated system can only transmit.

Definition 5.1 *Suppose S is a sequence of actions and $P \xrightarrow{s}$. Then s is a* computation *of P if every element of s is a transmission.* ∎

So the only computations of the *CATSYSTEM* are *meiosis*!, *meiosis*!*miao*! and *meiosis*!*miao*!*ha*!. *MEIOSIS* will not change state by herself.

The reads and discards of a subsystem are interesting only if they contribute to a transmission by the whole (isolated) system.

5.2 Testing equivalence

We add a special signal *succ* to the signal set $\mathcal{S}$. Let $SUCC \stackrel{\text{def}}{=} succ!\,\mathbf{0}$. *Testers* are now just CBS agents. We test an agent by putting it in parallel with a tester, and observing the computations of this composite system. A *successful computation* is one that includes *succ*. Note that we require the *succ* to appear in a finite number of steps. If s is a successful computation, every starting subsequence s' of s is said to have a *successful extension*.

Definition 5.2 *We say that an agent P* may pass *a test T if $P \mid T$ has a successful computation. P* must pass *T if every computation s of $P \mid T$ is either successful or has a successful extension.*

We abbreviate "must pass" by *must* and "may pass" by *may*. *P must T* implies *P may T*. We say *P fails T* if not *P may T*, i.e., if $P \mid T$ has no successful computation.

	$P \quad Q$	Tester T	P's result	Q's result
1.	$a!\,\mathbf{0} \not\simeq b!\,\mathbf{0}$	$a?\,SUCC$	*must*	*fails*
2.	$a?\,\mathbf{0} \simeq b?\,\mathbf{0}$			
3.	$a?\,c!\,\mathbf{0} \not\simeq b?\,c!\,\mathbf{0}$	$a!\,c?\,SUCC$	*must*	*fails*
		$b!\,c?\,SUCC$	*fails*	*must*
4.	$a?\,\mathbf{0} + c!\,\mathbf{0} \not\simeq b?\,\mathbf{0} + c!\,\mathbf{0}$	$a!\,c?\,SUCC$	*fails*	*may*
5.	$a!\,(b!\,\mathbf{0} + c!\,\mathbf{0}) \simeq a!\,b!\,\mathbf{0} + a!\,c!\,\mathbf{0}$			
6.	$a!\,(b?\,\mathbf{0} + c!\,\mathbf{0}) \not\simeq a!\,b?\,\mathbf{0} + a!\,c!\,\mathbf{0}$	$a?\,c?\,SUCC$	*must*	*may*

Table 5: Some testing results

Thus in the *CATSYSTEM*, the subsystem *MEIOSIS* $\mid$ *FRIEND must OWNER*. We could also say *OWNER must MEIOSIS* $\mid$ *FRIEND*, or that *MEIOSIS must OWNER* $\mid$ *FRIEND*, or that *CATSYSTEM must* **0**, and so on. It is only convention who is the "process" and who the tester.

We define agents P and Q to be *equivalent*, $P \simeq Q$, if no test can tell them apart.

Definition 5.3
$P \simeq_{may} Q$ *if* $\forall T$, P *may* T *iff* Q *may* T.
$P \simeq_{must} Q$ *if* $\forall T$, P *must* T *iff* Q *must* T.
$P \simeq Q$ *iff* $P \simeq_{may} Q$ *and* $P \simeq_{must} Q$.

Proposition 5.4 $\simeq$ *is substitutive wrt* $\mid$. *If* $P \simeq Q$, *then* $P \mid R \simeq Q \mid R$.

Proof: With any tester T, P and Q can be thought of as in the presence of a tester $T \mid R$. This tester cannot tell them apart, since $P \simeq Q$. But the success states are in T, which thinks it is testing $P \mid R$ and $Q \mid R$. That is, $P \mid R$ and $Q \mid R$ cannot be told apart. ∎

To prove agents unequal, we have to find a test to distinguish them. There are some examples in Table 5. (4) shows that $\simeq$ is not respected by $+$ and so $\simeq$ is not a congruence.

To prove agents equal is hard; we have to show that no test can tell them apart. But we can informally see the equalities in the table. Agents that never transmit (2) cannot be told apart by any test, since they never affect the tester.

In (5), our experience with CCS leads us to expect that P *must* $a?\,b?\,SUCC$, while Q *may* $a?\,b?\,SUCC$. Indeed Q can fail this test by choosing the $a!\,c!$ branch, but so can P! Whatever any tester does, all it can tell about these agents is that they will do $a!\,b!$ or $a!\,c!$, deciding internally which.

$$
\begin{array}{rcl}
T_i(s_l, s_r) & \stackrel{\text{def}}{=} & e_l?\, T_i(a, s_r) + e_r?\, T_i(s_l, a) + t_l?\, T_i(p, s_r) + t_r?\, T_i(s_l, p) + h_i!\, H_i(s_l, s_r) \\
\text{if not } s_l = s_r = p, & & \\
H_i(s_l, s_r) & \stackrel{\text{def}}{=} & e_l?\, H_i(a, s_r) + e_r?\, H_i(s_l, a) + t_l?\, H_i(p, s_r) + t_r?\, H_i(s_l, p) \\
H_i(p, p) & \stackrel{\text{def}}{=} & e_l?\, H_i(a, p) + e_r?\, H_i(p, a) + e_i!\, E_i(p, p) \\
E_i(s_l, s_r) & \stackrel{\text{def}}{=} & t_i!\, T_i(s_l, s_r) \\
SYS & \stackrel{\text{def}}{=} & \prod_{i=0}^{4} T_i(p, p)\,[\phi_i]
\end{array}
$$

where ϕ_i renames subscripts l to $i-1 \bmod 5$, and r to $i+1 \bmod 5$.

$$
\begin{array}{rcl}
\text{if not } s_l = s_r = a, & & \\
S_{i,j}(s_i, s_j) & \stackrel{\text{def}}{=} & e_i?\, S_{i,j}(a, s_j) + e_j?\, S_{i,j}(s_i, a) + t_i?\, S_{i,j}(p, s_j) + t_j?\, S_{i,j}(s_i, p) \\
S_{i,j}(a, a) & \stackrel{\text{def}}{=} & SUCC
\end{array}
$$

Table 6: Conversing philosophers

Proposition 5.5 *$P \sim Q$ implies $P \simeq Q$.*

Proof: $P \sim Q$ implies $P \,|\, T \sim Q \,|\, T$ for any tester T. The rest follows easily. ∎

Thus the strong bisimulation laws previously listed hold also for $\simeq$.

5.3 Example: Towards conversing philosophers

The dining philosophers problem usually concerns philosophers who eat, think, and have lively discussions with forks. We use a more abstract version [CM88]. There are five philosophers around a table. Each has three states, thinking, hungry and eating, and cycles between these states. A transition from hungry to eating is permitted only if neighbours are not eating; so neighbours do not eat simultaneously.

Table 6 shows how to model this in CBS. The ith philosopher has states T_i, H_i and E_i, with moves $t_i!$, $h_i!$, $e_i!$ leading to them. The neighbours use the moves e_l, e_r, t_l and t_r. Their states are a (for "active", i.e., eating) and p (for "passive", i.e., thinking or hungry).

$S_{i,j}$ is a tester that checks whether philosophers i and j ever eat simultaneously. The requirement that neighbours not eat simultaneously is formulated as the test $S = \prod_{i=0}^{4} S_{i,i+1 \bmod 5}(p, p)$, which we want SYS to *fail*.

Write $R_i(s_l, s_r)$ for the subsystem $T_2(s_l, s_r) \,|\, S_{2,3}(s_l, s_r) \,|\, T_3(s_l, s_r)$. Then $R_i(a, a)$ is the only state of this form that has a *succ*!. We can check that there is no move to this state from any of the three states $R_i(a, p)$, $R_i(p, p)$ or $R_i(p, a)$. It is now easy to see by induction that $R_i(p, p)$ only leads to one of these three states. So no computation of $R_i(p, p)$ can be successful.

Now consider $SYS \,|\, S$. The S_i do not transmit anything except for the final *succ*. Their presence does not affect the P_i, and each S_i is affected only by "its" pair of philosophers.

None of the S_i therefore get to $SUCC$, and SYS *fails* S.

Conclusions: By formulating requirements as tests, we can do some work even without proving equivalence. CBS seems more like a shared-state system than a distributed system. The broadcast is really useful; each e_i above was received by four other agents, two testers and two philosophers.

5.4 Silent actions and testing

1. $\tau!\, a?\, b!\, \mathbf{0} \not\simeq a?\, b!\, \mathbf{0}$
2. $\tau!\, b!\, \mathbf{0} \simeq b!\, \mathbf{0}$
   $a!\, \tau!\, b!\, \mathbf{0} \simeq a!\, b!\, \mathbf{0}$
   $a?\, \tau!\, b!\, \mathbf{0} \simeq a?\, b!\, \mathbf{0}$
3. $\tau!\, P + P \simeq P$

Table 7: Some testing results with $\tau!$

(1) in Table 7 shows that $\tau!$ can cause timing problems. Let $T \stackrel{\text{def}}{=} a!\, b?\, SUCC$. Then $a?\, b!\, \mathbf{0}$ *must* T, but $\tau!\, a?\, b!\, \mathbf{0}$ *may* T. Note that $c!\, a?\, b!\, \mathbf{0}$ could fail the test for exactly the same reason; only the fact that the τ is silent could mislead us to expect equivalence.

However, all is not lost, as (2) shows. Any tester will get a b from both agents. It cannot detect the τ. Nor can the tester do a timeout; to do this, we need to be able to do a clock tick that can be multiplied with any other transmission without afffecting the latter. There is no such action.

The corollaries to (2) are absorption laws. The latter says that internal computation between a service request a and provision b does not affect the user.

6 Discussion

6.1 Autonomy

An autonomous action should absorb no information. Receptions absorb information, and so are controllable: delayed till the information is available. The autonomy of transmissions does not follow from the transition system itself, but is imposed from outside by the definition 5.1 of computation. However, this definition is not arbitrary.

Scoping Computation should be defined so that scoping restricts controllable actions and hides autonomous ones.

Ranking For any action ν, $a!\, o\nu$ is either undefined or equal to $a!$. We say $a!$ is *perfect*, or has the highest ranking. An agent doing a perfect action is not affected by synchronisation, and absorbs no information. Perfect actions are natural choices for autonomy.

Suppose we defined reads to be autonomous instead of transmits. Then $a?\,Q$ could do an $a?$ in any context, thus absorbing no information. The perfection of $a!$ means $a!\,P$ cannot absorb any either. Thus there would be no communication at all. A related problem is that $a!\,P$ would be unable to act in any context. Perfect actions offer more progress. Speech takes hearing along with it, not the other way around.

Audibility $\tau!$ and $a!$ are both autonomous, but $a!$ is audible, while $\tau!$ is silent. CBS captures this by the fact that $\tau!$ can only synchronise with discards, which never absorb information.

6.2 Differences between the CBS and CCS models

τ vs. $\tau!$ Some properties of actions in CBS:

internal actions $\tau!$ { result of communication; perfect (highest ranking); autonomous; cannot be restricted, only hidden } } audible actions $a!$

internal actions $\tau!$ { silent; indirectly observable } } controlled actions $a?$; abstracted from in $\simeq$

In a CCS handshake, the two-way information flow means both actions are audible as well as controlled. Both are low ranking. τ is high ranking, and therefore autonomous. It is the linchpin in CCS. It has all the properties in the list, while no other action has any of them! CBS shows that some decoupling of these properties is possible; how much further we can go (in other calculi) is an interesting question.

Some differences between the calculi are summarised in Table 8.

6.3 Synchrony or asynchrony?

[BKT84] classify models of concurrency into synchronous or asynchronous along two dimensions, cooperation and communication. CCS has asynchronous cooperation and synchronous communication. CBS is hard to classify.

Cooperation Synchronous cooperation is typified by SCCS, MEIJE and ASSCS: every agent has to act at every step. Discards give CBS a spurious synchronous appearance, since all agents have to "act" together on every step. However, CBS seems asynchronous at a higher level, when we ignore discards. I would classify cooperation in CBS as asynchronous, since transmissions cannot be combined, and because we cannot program a timeout in CBS. But since $\tau!\,P \not\simeq P$, CBS is still "more synchronous" than CCS!

	CCS	CBS
Communication	Inaudible to others	Audible to others
Observation	Affects observed agent	Observed agent unaffected
Input=Output?	In pure CCS	Not even in pure CBS
Silent actions	Only one, τ. Can be indirectly detected.	$a?$, $b?$, $c:$, and $\tau!$ are different silent actions. $a?$, $b?$ and $\tau!$ can be indirectly detected, but not c:
Abstraction	From τ. Because it is internal? autonomous? silent?	From silent actions. We have an abstracting equivalence even without $\tau!$; $a?\,\mathbf{0} \simeq b?\,\mathbf{0}$
τ in practice	Central. Any interesting computation involves τ.	Much less important. Meaningful examples without $\tau!$.
Interleaving	Because an observer can hear only one thing at a time.	Because of the single channel. Same order for all observers, a meaningless remark in CCS.

Table 8: Some differences between CCS and CBS

Communication In synchronous communication, actions communicate only if performed simultaneously. CBS would thus appear to use synchronous communication. But [SNP87] argue that broadcast communication is asynchronous since the sender of a message does not wait for the receiver. CBS (everyday speech!) thus uses synchronous communication by one criterion, and asynchronous by another.

CBS makes more distinctions than truly asynchronous calculi. Those in [JJH90] and [JU90] have the results $a!\,b!\,P \simeq b!\,a!\,P$ and $a?\,b?\,P \simeq b?\,a?\,P$, neither of which hold in CBS.

6.4 Alternative design choices

Without receptions We could have just one sort of transition and rules of the form

$$\frac{Q \xrightarrow{a!} Q'}{a?\,P \,\big|\, Q \xrightarrow{a!} P \,\big|\, Q'}$$

Such rules mix a semantic requirement on one component with a syntactic one on the other. More importantly, they do not reflect the experience of the listener. Hearing is silent, but that is no reason to deny that it is an activity.

No discards Agents would be always prepared to listen, and explicitly discard uninteresting messages. This is more natural with buffered broadcast[Bro88], where uninteresting buffers need never be looked at.

More parallelism? In this paper, even a $\tau!$ in a subsystem prevents any other communication in the whole system. We can relax this, but not to the extent of $h! \circ \tau! = h!$. Then $h?\, H \backslash \{c\}$ could miss a $h!$ as follows.

$$\dfrac{\dfrac{\dfrac{c!\, C \xrightarrow{c!} C \quad h?\, H \xrightarrow{c:} h?\, H}{c!\, C \,\big|\, h?\, H \xrightarrow{c!} C \,\big|\, h?\, H}}{\left(c!\, C \,\big|\, h?\, H\right) \backslash \{c\} \xrightarrow{\tau!} \left(C \,\big|\, h?\, H\right) \backslash \{c\}} \; c \in \{c\} \qquad F \xrightarrow{h!} F'}{\left(c!\, C \,\big|\, h?\, H\right) \backslash \{c\} \,\big|\, F \xrightarrow{h!} \left(C \,\big|\, h?\, H\right) \backslash \{c\} \,\big|\, F'}$$

$\tau! \circ \tau! = \tau!$ is the extent to which we could allow more parallelism, allowing subsystems to proceed independently. The law $\left(P \,\big|\, Q\right) \backslash \mathcal{N} \sim (P \backslash \mathcal{N}) \,\big|\, (Q \backslash \mathcal{N})$ if $\mathcal{L}(P) \cap \mathcal{L}(Q) \cap \mathcal{N} = \emptyset$ would then no longer hold, for $\left(a!\, \mathbf{0} \,\big|\, b!\, \mathbf{0}\right) \backslash \{a, b\}$ would still need two steps to move to $\mathbf{0}$, while $a!\, \mathbf{0} \backslash \{a, b\} \,\big|\, b!\, \mathbf{0} \backslash \{a, b\}$ could do so in one. The law would hold for $\simeq$, however.

6.5 Related variations on CCS and CSP

That input and output are different affects us perhaps more than the multiple receivers for each communication. Most of the theory of process calculi abstracts from input and output to uninterpreted actions. It will therefore not be directly applicable to CBS.

Even in handshake communication, value-passing causally distinguishes sender from hearer. So theories of asynchronous communication [Jon90] [JJH90] [JU90] are of interest, as are theories of value-passing [HI89], and [Hen90] which adds value-passing to CSP. This last has a notion of broadcasting, with $a! \circ a? = a!$. The intuition is very different from ours, however, because transmitters can synchronise, but listeners may not.

7 Conclusions and future work

This is a working paper. Much needs to be done before any useful work can be done with CBS: some proof method for testing equivalence, as well as work on a testing congruence, for example. All process calculus can be applied to CBS, even if only to raise questions.

We have achieved an intuitively appealing model of communication, based on everyday speech. We have captured this in a calculus. The transition system seems well behaved. CBS has cast light on issues such as autonomy and asynchrony, and so on CCS as well. An intuitively reasonable testing equivalence has been defined, which shows some interesting results. Tests are just agents, and can be used as specifications.

8 Acknowledgements

Alan Jeffrey, Jenny Petersson, Uno Holmer, Wang Yi and others at Chalmers have been constant sources of help and encouragement. CBS began with a suggestion by Toivo Aasma two years ago. Many of the ideas here were first worked out at the Indian Institute of Science, Bangalore, in early 1990. I am grateful to Ranjani and other colleagues there for interesting discussions. Laura Lockwood suggested the airport example. This work was presented at the Nordic Workshop at Aalborg in October 1990, and discussed informally with participants at the Workshop on Semantics of Concurrency at Leicester, 1990, and at CONCUR'90 at Amsterdam. The referees gave detailed and helpful comments. I am very grateful to all for their criticisms and encouragement.

References

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

[BA90] M. Ben-Ari. *Principles of Concurrent and Distributed Programming.* Prentice-Hall, 1990.

[BCG86] G. Berry, P. Couronné, and G. Gonthier. Synchronous programming of reactive systems: An introduction to ESTEREL. Technical Report 647, INRIA, 1986.

[BIM88] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. In *15th Symposium on Principles of Programming Languages.* ACM, 1988.

[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.

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

[CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design—A Foundation.* Addison-Wesley, 1988.

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

[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.

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

[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.

[HI89] Matthew Hennessy and Anna Ingolfsdottir. A theory of communicating processes with value-passing. Technical Report 3/89, University of Sussex, 1989. Also presented at ICALP 90.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

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

[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.

[Jon90] Bengt Jonsson. A hierarchy of compositional models of I/O automata. Technical report, Swedish Institute of Computer Science, 1990.

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

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

[Mil80] Robin Milner. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science. Springer-Verlag, 1980.

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

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

[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.

[VG89] Frits Vaandrager and Jan Groote. Structured operational semantics and bisimulation as a congruence. In *Springer Verlag LNCS 372*, pages 423–438. ICALP, 1989.

[Win84] Glynn Winskel. Synchronization trees. *Theoretical Computer Science*, 34:33–82, 1984.