Acta Informatica 29, 737–760 (1992)

An efficiency preorder for processes

S. Arun-Kumar[1] **and M. Hennessy**[2]

[1] Department of Computer Science and Engineering, Indian Institute of Technology, Hauz Khas, New Delhi 110 016, India
[2] School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton BN1 9QH, England

Received January 14, 1991 / July 14, 1992

Abstract. A simple efficiency preorder for CCS processes is introduced in which $p \lesssim q$ means that q is at least as fast as p, or more generally, p uses at least as much resources as q. It is shown to be preserved by all CCS contexts except summation and it is used to analyse a non-trivial example: two different implementations of a bounded buffer. Finally we give a sound and complete proof system for finite processes.

1. Introduction

A large number of behavioural equivalences for process description languages have been studied in recent years [11, 9, 6]. If $\approx$ is such an equivalence then $p \approx q$ means intuitively that p and q offer essentially the same behaviour to the environment. When comparing the behaviour of processes, many of these equivalences, often called asynchronous equivalences, do not take timing considerations into account; internal actions are considered to be instantaneous. On the other hand there are many applications for which it would be reasonable to have the notion of behaviour include at least some aspects of time. The description of real-time systems is the most obvious candidate and the proper functioning of many communication protocols depend at least to some extent on the fine-tuning ot the relative timings of certain actions. This has led to the development of real-time versions of standard process description languages. Typical examples include "real-time LOTOS" [15] and timed CSP [14]. In these languages all actions have a specific time or duration associated with them and, roughly speaking, two processes are deemed to be equivalent if they

* Most of this work was done while the first author was at the University of Sussex and supported by SERC grant GR/D 97368 of the Science and Engineering Research Council of Great Britain.
** The second author would like to acknowledge the support of the ESPRIT II Basic Research Action Concur.

offer the same potential actions of the same duration at more or less the same time during a computation. Let us call this intuitive idea "real-time equivalence".

For many applications this treatment of time is much too detailed. One ends up being forced to carry out long and precise timing calculations, the details of which are often superfluous. Indeed, many descriptions which one feels should be equivalent are not so because the comparison is too fine-grained. We would like to develop methods of comparing processes which are finer than the asynchronous equivalences in that they take time into consideration but are not as restrictive as real-time equivalences.

In this paper (which is a revised version of [1]) we develop one such method. The idea is very simple and although it will not be universally applicable, we feel that it will be useful in many applications. We develop a preorder on process descriptions which has approximately the following meaning: $p \lesssim q$ if p and q are bisimulation equivalent and q is at least as fast as p.

So in this approach there is no assignment of absolute values to the actions or constructs of a process. Instead it is purely comparative; the comparison is made by assuming that all external actions take the same time and that internal actions take some indefinite but non-zero time. Hence if α and β are actions and 1 represents an internal move then it turns out that

$$\alpha.1.\beta.\mathbf{0} \lesssim \alpha.\beta.\mathbf{0}$$

but

$$\alpha.\beta.\mathbf{0} \not\lesssim \alpha.1.\beta.\mathbf{0}.$$

One obvious advantage of such a treatment as opposed to the usual real-time equivalences such as [14] and [5] is that "1" may be used not just as a unit or measure of time but to denote some more general quantities such as "energy consumed in a computation" or the "complexity of communication and synchronization" as well.

We will show in this paper that a mathematically tractable behavioural theory of processes based on these ideas can be developed. It is applied to the standard version of CCS although in principle there is no reason why it should not be equally to more general languages with real-time features.

In Sect. 2 we define the syntax and operational semantics of CCS. This is completely standard and may be omitted by readers familiar with CCS. In Sect. 3 we show how to modify the usual recursive definition of bisimulation equivalence [11], so as to obtain $\lesssim$. Roughly speaking for $p \lesssim q$ to hold, every "weak arrow", $\stackrel{\alpha}{\Rightarrow}$, from p must be matched by a corresponding "weak arrow" from q which performs at most the same number of internal moves and conversely, every "weak arrow" from q must be matched by a "weak arrow" from p which uses at least as many internal moves. Alternative and more useful formulations of $\lesssim$ are also given. We develop most of these concepts by confining ourselves to "pure CCS", where no values are passed between processes. The extension to value-passing may be done in the usual fashion. We then investigate the algebraic properties of $\lesssim$. For the same reasons as for observational equivalence (see [11]) it is not preserved by the "+" operation of CCS, but we can apply the usual method to overcome this problem. We show that the resulting relation is preserved by all the operators of CCS including recur-

sion. We also justify fixpoint induction. In Sect. 4, we consider an example which examines different methods of implementing a FIFO buffer. In Sect. 5 we show how a complete proof system may be obtained for finite processes. This involves modifying the equations in [7] but it appears that an extra proof rule is also necessary. The paper ends with a brief comparison with other related work in the literature.

2. CCS and labelled transition systems

2.1 Syntax and operational semantics of CCS

Let A and $\bar{A}$ (the complement of A) be infinite disjoint sets in bijection under the complementation operation "$^-$". Then $V=A\cup\bar{A}$ is the set of *ports* and complementation is extended to the whole of V so that $\bar{\bar{\alpha}}=\alpha$ for all $\alpha\in V$. If D is any domain of values, $V\times D$ denotes the set of *visible actions* with "$^-$" extended to $V\times D$ in the usual way. That is, for $\alpha\in V$ and $d\in D$, $\alpha(d)$ will denote an element of $V\times D$ and $\bar{\alpha}(d)$ its complement. Let "1" be a special action called the *internal* (*silent* or *invisible*) action, such that $\bar{1}=1$. Then $A=(V\times D)\cup\{1\}$ is the set of all *actions*. In particular, when D is a single-valued domain we shall identify V with $V\times D$.

Unless otherwise mentioned, we use the following notational conventions. Typically $\alpha,\beta,\gamma,\ldots$ (suitably decorated) denote elements of V and $a,b,c,\ldots$ (suitably decorated) stand for (visible or invisible) actions. Let X be a set of process variable symbols and U a set of value variable symbols. Then $x,y,z,\ldots$ (possibly decorated) represent process variable names and $u,v,w,\ldots$ (possibly decorated) stand for value variable symbols. The language of CCS expressions is then given by the following BNF.

$$e ::= x\,|\,\Phi\,|\,\alpha(u).e\,|\,\bar{\alpha}(d).e\,|\,1.e\,|\,e+e\,|\,e|e\,|\,e\backslash L\,|\,e[h]\,|\,\mu_i\,\mathbf{x}{:}\mathbf{e},$$

where Φ is a constant (or 0-ary operator), $u\in U$, $d\in D$ and $L\subseteq V$, $\mathbf{x}$ and $\mathbf{e}$ denote vectors of process variables and process expressions respectively, and $\mu\mathbf{x}$: $\mathbf{e}$ denotes the solution of a system of recursive process equations, whose i-th component is $\mu_i\mathbf{x}$: $\mathbf{e}$. In the relabelling operation $e[h]$, h: $A\to A$ is the relabelling function such that $\overline{h(\alpha)}=h(\bar{\alpha})$ and $h(\alpha)\neq 1$ for all $\alpha\in V$ and $h(1)=1$.

Since our results are quite general enough to apply to many-valued domains, we will not explicitly specify elements of the domain or value variable names unless required in the context. Hence "$\alpha(u).e$" will often be written as "$\alpha.e$" if the value variable u is understood or unimportant. Similarly where actual values are unimportant, we shall use $\alpha,\beta,\gamma,\ldots$ (suitably decorated) to range over the domain $V\times D$ and $a,b,c,\ldots$ to range over the domain $V\times D$ and $a,b,c,\ldots$ to range over A.

Any term generated by the above BNF is called a *process expression* and E, ranged over by $e,f,g,\ldots$ (possibly decorated), denotes the set of process expressions. For any process expression e, FPV(e) is the set of free process variables in e. *Processes* are closed process expressions (i.e. expressions with no free process variables). $p,q,r,\ldots$ (possibly decorated) range over the set P of all processes.

We have defined a subset of CCS with value-passing that is sufficient for our purposes. A more complete language would have a conditional statement

and a "sublanguage" for value expressions. Since these extra features are not used in our examples we have excluded them for the sake of simplicity.

The operational semantics of the language is defined in terms of labelled transition systems in the usual fashion. Let $\langle E, A, \{\xrightarrow{a} | a \in A\}\rangle$ be a labelled transition system (LTS) where the transition relation $\rightarrow \subseteq E \times A \times E$ is the smallest relation satisfying the following axioms and rules for inference.

1. Prefix

1.1 $\alpha(u).e \xrightarrow{\alpha(d)} e\{d/u\}$, for all $d \in D$

1.2 $\bar{\alpha}(d).e \xrightarrow{\bar{\alpha}(d)} e$

1.3 $1.e \xrightarrow{1} e$

2. Summation

2.1 $e_1 \xrightarrow{a} e_1' \Rightarrow e_1 + e_2 \xrightarrow{a} e_1'$

2.2 $e_2 \xrightarrow{a} e_2' \Rightarrow e_1 + e_2 \xrightarrow{a} e_2'$

3. Composition

3.1 $e_1 \xrightarrow{a} e_1' \Rightarrow e_1 | e_2 \xrightarrow{a} e_1' | e_2$

3.2 $e_2 \xrightarrow{a} e_2' \Rightarrow e_1 | e_2 \xrightarrow{a} e_1 | e_2'$

3.3 $e_1 \xrightarrow{\alpha} e_1', e_2 \xrightarrow{\bar{\alpha}} e_2' \Rightarrow e_1 | e_2 \xrightarrow{1} e_1' | e_2'$

4. Hiding

$$e \xrightarrow{a} e', a, \bar{a} \notin L \Rightarrow e \backslash L \xrightarrow{a} e' \backslash L$$

5. Relabelling

$$e \xrightarrow{a} e' \Rightarrow e[h] \xrightarrow{h(a)} e'[h]$$

6. Recursion

$$e_i\{\mu \mathbf{x}: \mathbf{e}/\mathbf{x}\} \xrightarrow{a} e' \Rightarrow \mu_i \mathbf{x}: \mathbf{e} \xrightarrow{a} e'$$

where "$f\{\mathbf{r}/\mathbf{x}\}$" denotes the syntactic substitution of all free occurrences of the variable $x_j (x_j \in \mathbf{x})$ in the expression f by $r_j (r_j \in \mathbf{r})$ for each j in the indexing set of the vectors $\mathbf{x}$ and $\mathbf{r}$.

2.2 Derived labelled transition systems

We may readily extend the notion of labelled transition systems to derived LTS (DLTS) and weak LTS (WLTS) such that a transition is over a sequence of actions.

Definition 2.1 A *derived LTS* is a structure $\langle E, A^*, \{\xrightarrow{s} | s \in A^*\}\rangle$, where $\rightarrow \subseteq E \times A^* \times E$ is the least relation satisfying the following conditions:

(i) $e \xrightarrow{\varepsilon} e$, for all $e \in E$, where ε is the empty sequence,

(ii) $e \xrightarrow{as} e'$ if for some $e_1 \in E$, $e \xrightarrow{a} e_1$ and $e_1 \xrightarrow{s} e'$.

Definition 2.2 A *weak LTS* is a structure of the form $\langle E, (V \times D)^*, \{\overset{s}{\Rightarrow} | s \in (V \times D)^*\}\rangle$, where for $s = \alpha_1 \ldots \alpha_n \in (V \times D)^*$, $e \overset{s}{\Rightarrow} e'$ iff there exists $t = 1^{m_0}\alpha_1 1^{m_1} \ldots 1^{m_{n-1}}\alpha_n 1^{m_n}$ in A^*, for $m_i \geqq 0$, $0 \leqq i \leqq n$, such that $e \xrightarrow{t} e'$.

For any $t \in A^*$, let $\hat{t}$ denote the sequence (of visible actions) obtained by deleting all occurrences of the internal action 1 from t. In the above definition $\hat{t} = s$. Following Milner we use the notation $p \Rightarrow p'$ to denote $p \xrightarrow{1}{}^* p'$ and $p \xrightarrow{1}{}^+ p'$ to denote $p \xrightarrow{1}{}^+ p'$.

Definition 2.3 Let $\leqq$ be the binary relation on A^* generated by the inequations $1s \leqq s$ and $s \leqq s$, i.e. $\leqq$ is closed under reflexivity, transitivity, and substitution under catenation contexts. An *extended action* is an element of A^* containing at most one visible action. The set EA of all extended actions is partially ordered by $\leqq$.

Note that the empty sequence and any finite sequence of internal actions belong to EA. In fact, $A \subseteq \mathrm{EA} \subseteq A^*$. It is also clear that $\leqq$ is antisymmetric and hence a partial order on A^* and EA.

In the next section we proceed with our analysis of the effect of this partial order on process behaviours. However to enable the reader to look at our definitions and theorems in the proper perspective, we also intersperse in our theory some results (stated without proof) due to Milner. The reader may consult [11] for the relevant proofs.

3. Weak bisimulations and efficiency prebisimulations

Definition 3.1 If $\mathrm{FPV}(e) \cup \mathrm{FPV}(f) \subseteq \mathbf{x}$ and $\lhd$ is a binary relation on processes, then $e \lhd f$ iff for every vector of processes $\mathbf{p}$, $e\{\mathbf{p}/\mathbf{x}\} \lhd f\{\mathbf{p}/\mathbf{x}\}$.

The above definition enables us to extend every behavioural relation on processes to process expressions. In the sequel we will assume that every behavioural relation that we define is thus extended. We now define the notion of weak bisimulations [10] and state some of its properties.

Definition 3.2 A binary relation $R \subseteq P \times P$ is a *weak bisimulation* (abbreviated to *wb*) if for every $\langle p, q\rangle \in R$ and $t \in (V \times D)^*$ the following conditions are satisfied.

WB1. $p \overset{t}{\Rightarrow} p' \Rightarrow \exists q' : q \overset{t}{\Rightarrow} q' \wedge p' \, R \, q'$

WB2. $q \overset{t}{\Rightarrow} q' \Rightarrow \exists p' : p \overset{t}{\Rightarrow} p' \wedge p' \, R \, q'$

Proposition 3.3 *1. If R_1 and R_2 are* wbs *then so is $R_1 \circ R_2$.*
2. The union of a family of wbs *is a* wb.
3. $\approx = \bigcup \{R \mid R$ is a wb$\}$ is the largest wb.
4. $p \approx q$ iff for some wb *R, pRq.*
5. $\approx$ is an equivalence relation on P. □

The relation $\approx$ is called observational equivalence.

Definition 3.4 A binary relation $R \subseteq P \times P$ is an *efficiency prebisimulation* (*ep* for short) if for every $\langle p, q \rangle \in R$ and $s, s' \in EA$ the following conditions are satisfied.

EP1. $p \xrightarrow{s} p' \Rightarrow \exists s' : s \leqq s' : \exists q' : q \xrightarrow{s'} q' \wedge p' R q'$

EP2. $q \xrightarrow{s'} q' \Rightarrow \exists s : s \leqq s' : \exists p' : p \xrightarrow{s} p' \wedge p' R q'$.

As in the case of weak bisimulation we may prove the following proposition. In addition, it is also easy to show that every efficiency prebisimulation is, in fact, a weak bisimulation.

Proposition 3.5 *1. Every efficiency prebisimulation is a weak bisimulation.*
2. If R_1 and R_2 are eps *then so is $R_1 \circ R_2$.*
3. The union of a family of eps *is an* ep.
4. $\lesssim = \bigcup \{R \mid$ is an ep$\}$ is the largest ep.
5. $p \lesssim q$ iff for some ep *R, pRq.*
6. $\lesssim$ is a preorder on P. □

We give below a simpler formulation of efficiency prebisimulations. It is also more convenient to use than Definition 3.4.

Proposition 3.6 *A binary relation $R \subseteq P \times P$ is an* ep *iff for every $\langle p, q \rangle \in R$, $\alpha \in V \times D$, $a \in A$, the following conditions are satisfied.*

EP'1. $p \xrightarrow{\alpha} p' \Rightarrow \exists q' : q \xrightarrow{\alpha} q' \wedge p' R q'$

EP'2. $p \xrightarrow{1} p' \Rightarrow p' R q \vee (\exists q' : q \xrightarrow{1} q' \wedge p' R q')$

EP'3. $q \xrightarrow{a} q' \Rightarrow \exists p' : p \overset{a}{\Rightarrow} p' \wedge p' R q'$

Proof. ($\Rightarrow$) It is easy to see that EP1 implies EP'1 because for $s = \alpha$, $s \leqq s'$ implies $s' = \alpha$. EP1 also implies EP'2 since for $s = 1$, $s \leqq s'$ implies either $s' = \varepsilon$ or $s' = 1$. Also it is obvious that EP2 implies EP'3 for $s' = \alpha \in V \times D$. For $s' = 1$, from EP2 we have $p \xrightarrow{s} p'$ for some $s \leqq s'$, that is, $p \overset{1}{\Rightarrow} p'$. Hence EP implies EP'.

($\Leftarrow$) Suppose R is a relation satisfying the conditions EP'. Let $\langle p, q\rangle \in R$ and for some $s \in \mathrm{EA}$, let $p \xrightarrow{s} p'$. We proceed by induction on the length of s. If $s = \varepsilon$ there is nothing to prove. Otherwise $s = a s_1$ for some $a \in A$ and $s_1 \in \mathrm{EA}$. There exists p_1 such that $p \xrightarrow{a} p_1 \xrightarrow{s_1} p'$. Now we have two cases to consider.

Case (*i*) $a = \alpha \in V \times D$. By EP'1 there is a q_1 such that $q \xrightarrow{\alpha} q_1$ and $p_1 R q_1$. By the induction hypothesis there exists $s_1' \in \mathrm{EA}$ and q' such that $s_1 \leqq s_1'$, $q_1 \xrightarrow{s_1'} q'$ and $p' R q'$. Letting $s' = \alpha s_1'$ it is easy to see that EP1 holds.

Case (*ii*) $a = 1$. Then either $\langle p_1, q\rangle \in R$ or there is a q_1 such that $q \xrightarrow{1} q_1$ and $p_1 R q_1$. In the latter case the proof proceeds as in case (i). In the former instance, again by the induction hypothesis there must exist s_1', $s_1 \leqq s_1'$ and q' such that $q \xrightarrow{s_1} q'$ and $p' R q'$. Letting $s' = s_1'$ we have $s = 1 s_1 \leqq s_1' = s'$ and EP1 follows.

It is obvious that EP2 follows from EP'3. □

As in the case of $\approx$ [11] we may show that $\lesssim$ is preserved by all operators except summation. As the following example shows, it is due to the preemptive power of the silent action that $\lesssim$ and $\approx$ are not preserved under summation.

Example 3.7 $1.\alpha.\mathbf{0} \lesssim \alpha.\mathbf{0}$ whereas $1.\alpha.\mathbf{0} + \beta.\mathbf{0} \not\lesssim \alpha.\mathbf{0} + \beta.\mathbf{0}$ because $1.\alpha.\mathbf{0} + \beta.\mathbf{0} \xrightarrow{1} \alpha.\mathbf{0}$ while $\alpha.\mathbf{0} + \beta.\mathbf{0} \xrightarrow{\varepsilon} \alpha.\mathbf{0} + \beta.\mathbf{0}$. For the same reasons $1.\alpha.\mathbf{0} \approx \alpha.\mathbf{0}$ holds but $1.\alpha.\mathbf{0} + \beta.\mathbf{0} \approx \alpha.\mathbf{0} + \beta.\mathbf{0}$ does not.

In order to obtain a precongruence we therefore follow the usual method of defining the largest precongruence contained in $\lesssim$. In the following lemma we state without proof the result that all the operators, except summation and recursion, preserve the preorder. It is also preserved under recursion but we omit the proof.

Lemma 3.8 *Let* $\lhd \in \{\lesssim, \approx\}$. *Then for all* $p, q \in P$, $p \lhd q$ *implies*
(a) $a.p \lhd a.q$ *for all* $a \in A$,
(b) $p|r \lhd q|r$ *and* $r|p \lhd r|q$ *for all* $r \in P$,
(c) $p \backslash L \lhd q \backslash L$ *for* $L \subseteq V$,
(d) $p[h] \lhd p[h]$ *for any relabelling function* h. □

It is also easy to verify that the 1-laws (called "τ-laws" in [11]) no longer hold symmetrically for the preorder $\lesssim$. We state these laws in the next lemma. In the sequel we always use $\lhd$ to denote either $\lesssim$ or $\approx$.

Lemma 3.9
1. $1.p \lhd p$
2. $1.p \lhd p + 1.p \lhd p$
3. $a.1.p \lhd a.p$

4. $a.(p+1.q) \lhd a.(p+1.q)+a.q$. □

This lemma may be proved for both $\lesssim$ and $\approx$ by constructing appropriate eps (note that it is not enough to construct wbs for the purpose of $\lesssim$, though every ep is a wb by Proposition 3.5.1).

Definition 3.10 For $\lhd \in \{\lesssim, \approx\}$, $p \lhd^+ q$ iff for some visible action α not occurring in p or q, $p+\alpha.\mathbf{0} \lhd q+\alpha.\mathbf{0}$.

Note that only a finite number of distinct action symbols appear in any process. Since Λ is infinite it is always possible to find a visible action that does not occur in a given process.

For reasons that will become clear in Theorem 3.15, the relations $\lesssim^+$ and $\approx^+$ defined above (Definition 3.10) are called *efficiency precongruence* and *observational congruence* respectively. In the next proposition we give a useful behavioural characterization of $\lesssim^+$.

Proposition 3.11

1. $p \lesssim^+ q$ *implies* $p \lesssim q$.

2. $p \lesssim^+ q$ *and* $p \xrightarrow{1} p'$ *implies* $\exists q' : q \xrightarrow{1} q' \wedge p' \lesssim q'$.

3. $p \lesssim^+ q$ *iff for every* $a \in A$, *the following conditions hold.*

EP$^+$1. $p \xrightarrow{a} p' \Rightarrow \exists q' : q \xrightarrow{a} q' \wedge p' \lesssim q'$

EP$^+$2. $q \xrightarrow{a} q' \Rightarrow \exists q' : p \overset{a}{\Rightarrow} p' \wedge p' \lesssim q'$

Proof. 1. Obvious.

2. Let $p \lesssim^+ q$. Then by Definition 3.10 $p+\alpha.\mathbf{0} \lesssim q+\alpha.\mathbf{0}$, for some visible action α that does not occur in p or q. If $p \xrightarrow{1} p'$ then $p+\alpha.\mathbf{0} \xrightarrow{1} p'$. From condition EP′2 (Proposition 3.6) $p' \lesssim q+\alpha.\mathbf{0}$ or for some q', $q+\alpha.\mathbf{0} \xrightarrow{1} q'$ and $p' \lesssim q'$. But $p' \not\lesssim q+\alpha.\mathbf{0}$ since $q+\alpha.\mathbf{0} \xrightarrow{\alpha} \mathbf{0}$, whereas p' can never perform an α action. Hence there must be a q' such that $q \xrightarrow{1} q'$ and $p' \lesssim q'$.

3. ($\Rightarrow$) Follows from parts 1 and 2.

($\Leftarrow$) It suffices to show from EP$^+$ that $p+\alpha.\mathbf{0} \lesssim q+\alpha.\mathbf{0}$. It is easy to show from EP$^+$ that EP′1 and EP′3 are satisfied by the pair $\langle p+\alpha.\mathbf{0}, q+\alpha.\mathbf{0} \rangle$. Let $p+\alpha.\mathbf{0} \xrightarrow{1} p'$, then since $\alpha \in V \times D$ we have $p \xrightarrow{1} p'$ and from EP$^+$1 it follows that for some q', $q \xrightarrow{1} q'$ and $p' \lesssim q'$. Hence $q+\alpha.\mathbf{0} \xrightarrow{1} q'$ and $p' \lesssim q'$ which proves EP′2. □

A similar behavioural characterization exists for $\approx^+$ which we merely reproduce without proof [11].

Proposition 3.12 *$p \approx^+ q$ iff for every $a \in A$, the following conditions are satisfied.*

WB$^+$1. $p \xrightarrow{a} p' \Rightarrow \exists q' : q \stackrel{a}{\Rightarrow} q' \wedge p' \approx q'$

WB$^+$2. $q \xrightarrow{a} q' \Rightarrow \exists p' : p \stackrel{a}{\Rightarrow} p' \wedge p' \approx q'$ □

It is then a simple matter to prove the following proposition for $\lhd^+ \in \{\lesssim^+, \approx^+\}$. We leave the proof to the reader.

Proposition 3.13 *$\lhd^+$ is preserved under prefix, summation, composition, hiding and relabelling.*

To show that $\lhd^+$ is a precongruence it is now only necessary to prove that it is preserved under recursion.

Lemma 3.14 *Let* **e** *and* **f** *be process expressions such that* $\mathbf{e} \lhd^+ \mathbf{f}$ *and* $FPV(\mathbf{e}) \cup FPV(\mathbf{f}) \subseteq \mathbf{x}$. *Then* $\mu_i \mathbf{x} : \mathbf{e} \lhd^+ \mu_i \mathbf{x} : \mathbf{f}$.

Proof outline. This was originally proved in [11] for $\approx^+$ and the same proof may be easily adapted for $\lhd^+$. We give an outline for the case of e and f having at most one free variable, say x. That is, we prove that $e \lhd^+ f$ implies $p \lhd^+ q$ where $p \equiv \mu x : e$ and $q \equiv \mu x : f$. Now consider the relation

$$R = \{\langle g\{p/x\}, g\{q/x\}\rangle \mid g \in E, \mathrm{FPV}(g) \subseteq \{x\}\}.$$

By induction on the depth of the inferences $g\{p/x\} \xrightarrow{a} p'$ and $g\{q/x\} \xrightarrow{a} q'$ respectively, one may show that R satisfies the following conditions for all $a \in A$.

(i) $g\{p/x\} \xrightarrow{a} p' \Rightarrow \exists q' : g\{q/x\} \xrightarrow{a} q' \wedge p' R \circ \lhd q'$

(ii) $g\{q/x\} \xrightarrow{a} q' \Rightarrow \exists p' : g\{p/x\} \xrightarrow{a} p' \wedge p' \lhd \circ R q'$

From (i) and (ii) we may show that $\lhd \circ R \circ \lhd$ is an ep if $\lhd = \lesssim$ and a wb if $\lhd = \approx$, and since $R \subseteq \lhd \circ R \circ \lhd$ it follows that $R \subseteq \lhd$. Letting $g \equiv x$ it is clear that $\langle p, q\rangle \in R$ and hence $p \lhd q$. But from (i) and (ii) above and the conditions EP$^+$ in Proposition 3.11.3, we obtain the stronger result $p \lhd^+ q$. □

Theorem 3.15 *$\lhd^+$ is a precongruence on CCS process expressions.*

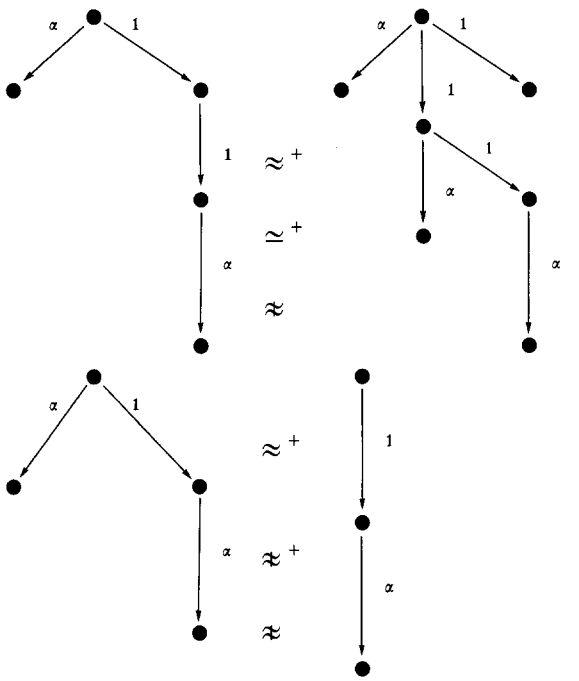
Proof. Follows from Proposition 3.13 and Lemma 3.14. □

Definition 3.16 The kernel of $\lesssim^+$, defined by $\simeq^+ = \lesssim^+ \cap \gtrsim^+$, where $\gtrsim^+$ denotes the converse of $\lesssim^+$, is called *efficiency congruence.*

We will also have occasion to refer to another relation which we briefly describe here. If Definition 3.2 were strengthened by letting $t \in A^*$ and replacing "$\stackrel{t}{\Rightarrow}$" everywhere by "$\xrightarrow{t}$" in the conditions WB, then the relation obtained is a

strong bisimulation and $\sim$, called *strong bisimulation congruence*, denotes the largest strong bisimulation (for details see [11]). It turns out that efficiency congruence strictly contains strong congruence, and is, in turn, strictly contained in observational congruence i.e., $\sim \subset \simeq^+ \subset \approx^+$. The following example makes clear the difference between the three relations.

Example 3.17.



4. An example: The (N + 2)-buffer

In this example we use the value-passing version to specify a bounded buffer of capacity $N+2$, $N>0$, and compare two different implementations of the specification. The problem may be stated informally as follows. Values are to be accepted from a process SOURCE and delivered *in order* to another process DEST independent of the speeds of execution of SOURCE and DEST. We give three different processes which carry out this task and which are strictly ordered with respect to $\lesssim$.

Let D be a value domain and for $n \geqq 0$, let D^n denote the set of strings of length n and $D_n = \bigcup \{D^k \mid 0 \leqq k \leqq n\}$ the set of strings of length at most n. The state of the buffer at any instant of its execution is determined by the string it contains. The state space of the buffer is given by D_{N+2}. We may then formally specify the behaviour of the buffer by the CCS process FIFO(t),

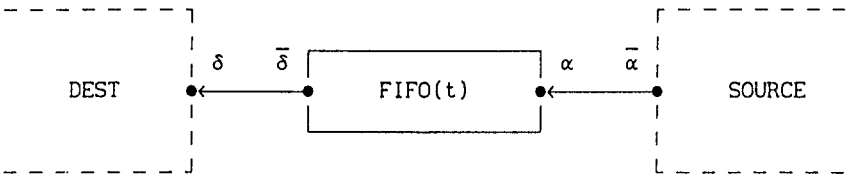


Fig. 1. Specification of buffer

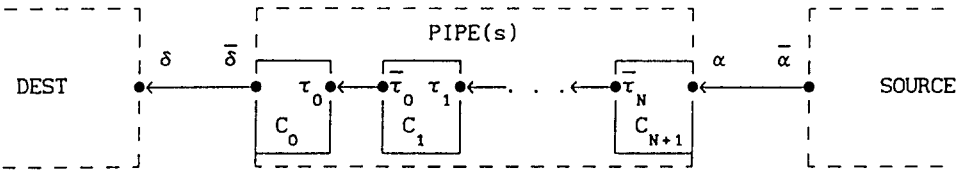


Fig. 2. First implementation of buffer

where $t \in D_{N+2}$, $a, d \in D$, α is the input port and $\bar{\delta}$ is the output port as shown in Fig. 1.

$$\text{FIFO}(t) \equiv \begin{cases} \alpha(a).\text{FIFO}(a) & \text{if } t = \varepsilon \ (\text{or } |t| = 0) \\ \bar{\delta}(d).\text{FIFO}(t_1) + \alpha(a).\text{FIFO}(ta) & \text{if } t = dt_1 \text{ and } 0 < |t| \leqq N+1 \\ \bar{\delta}(d).\text{FIFO}(t_1) & \text{if } t = dt_1 \text{ and } |t| = N+2. \end{cases}$$

Hence for any string t, FIFO(t) has the following transitions.

$$\text{FIFO}(t) \xrightarrow{\alpha(a)} \text{FIFO}(ta) \quad \text{if } 0 \leqq |t| < N+2$$

$$\text{FIFO}(t) \xrightarrow{\bar{\delta}(d)} \text{FIFO}(t_1) \quad \text{if } 0 < |t| \leqq N+2 \text{ and } t = dt_1.$$

Throughout this section we shall use the word "implementation" to refer to any CCS process that is observationally congruent to FIFO(ε). We shall also override some of the notational conventions specified in Sect. 2.1 as regards variables and values.

The process PIPE. A simple implementation of this buffer is in terms of cells, where a cell is an elementary $i-o$ device that may be defined as $C \equiv \iota(x).C'(x)$ and $C'(x) \equiv \bar{o}(x).C$. For the sake of uniformity we denote the state C as $C(\perp)$ and $C'(d)$ ad $C(d)$. A buffer of capacity $N+2$ may be created simply by connecting cells end to end and relabelling the ports appropriately (see Fig. 2.). Therefore we have

$$\begin{aligned} C_0(x) &\equiv C(x)[\tau_0/\iota, \delta/o] \\ C_j(x) &\equiv C(x)[\tau_j/\iota, \tau_{j-1}/o] \quad \text{for } 0 < j \leqq N \\ C_{N+1}(x) &\equiv C(x)[\alpha/\iota, \tau_N/o] \end{aligned}$$

for all $x \in D_\perp$, where $D_\perp = D \cup \{\perp\}$. Then

$$\text{PIPE}(s) \equiv (\Gamma\{C_j(x_j) \mid 0 \leqq j \leqq N+1\}) \setminus \{\tau_k \mid 1 \leqq k \leqq N+1\},$$

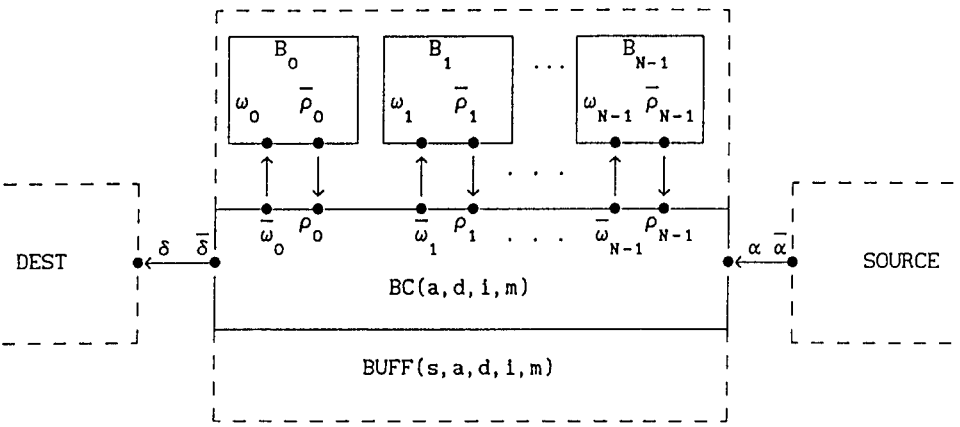


Fig. 3. Second implementation of buffer

where "Γ" denotes the composition of a set of processes and $s=x_0 \ldots x_{N+1}$, $s \in D_\perp^{N+2}$. The transitions of PIPE(s) are

$$\text{PIPE}(s' \perp) \xrightarrow{\alpha(a)} \text{PIPE}(s' a) \quad \text{where } s' \in D_\perp^{N+1}, a \in D$$

$$\text{PIPE}(d s') \xrightarrow{\bar{\delta}(d)} \text{PIPE}(\perp s') \quad \text{where } s' \in D_\perp^{N+1}, d \in D$$

$$\text{PIPE}(s' \perp v s'') \xrightarrow{1} \text{PIPE}(s' v \perp s'') \quad \text{where } s' s'' \in D_\perp^N, v \in D.$$

To prove that PIPE is an implementation of FIFO, it is sufficient to display a weak bisimulation between their state spaces. However it is possible to prove the stronger result given in the following lemma, whose detailed proof we leave to the interested reader.

Proposition 4.1 *Let* $f: D_\perp^{N+2} \to D_{N+2}$ *be the function which strips off all occurrences of "*$\perp$*" from any string in* $D_\perp^{N+2}$. *Then*
1. $R = \{\langle \text{PIPE}(s), \text{FIFO}(f(s)) \rangle | s \in D_\perp^{N+2}\}$ *is an* ep.
2. $\text{PIPE}(\perp^{N+2}) \lesssim^+ \text{FIFO}(\varepsilon)$.

Proof Outline. 1. Clearly f is surjective, so every possible state of PIPE and FIFO is present in R. Then it is a simple matter to show from the transitions of PIPE and FIFO that R is indeed an ep.
2. For $t=\varepsilon$ and $s=\perp^{N+2}$, it is clear that $f(s)=\varepsilon$ and also $\text{PIPE}(\perp^{N+2})$ has no silent transitions. Then PIPE(s) and FIFO(t) satisfy the conditions EP^+ of Proposition 3.11.3. Hence $\text{PIPE}(\perp^{N+2}) \lesssim^+ \text{FIFO}(\varepsilon)$. □

The process BUFF. Now consider a different implemetation of the buffer, in which N cells are used (numbered from 0 to $N-1$) simply as storage and each cell interacts only with a centralized buffer controller which may store an additional two values. We define the store as follows.

$$B_j(x) \equiv C(x)[\omega_j/\imath, \rho_j/o], \quad \text{for } 0 \leqq j < N$$
$$\text{MEM}(s) \equiv \Gamma\{B_j(x_j) | 0 \leqq j < N\},$$

where $s=x_0 \ldots x_{N-1}$ and $x_j \in D_\perp$, is the state of cell B_j for $0 \leqq j < N$.

The buffer controller, BC, whose schematic design is shown in Fig. 3, may store at most two values at any instant. It performs the following functions.

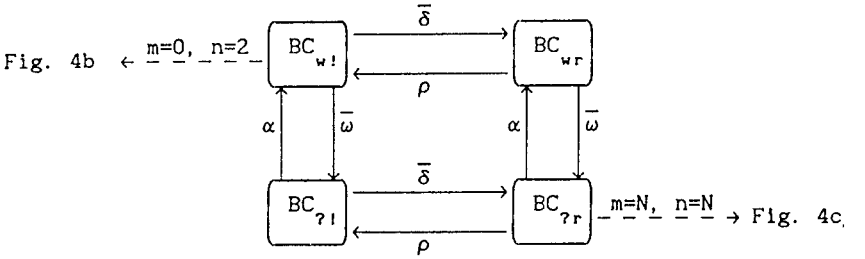


Fig. 4a. Transition diagram of BC when MEM is neither full nor empty.

(a) it *accepts* a value from SOURCE through the port α, and does not accept another till the first one is stored in one of the N cells.
(b) it *writes* the most recently accepted value in the first available empty cell, say B_j, through the port $\bar{\omega}_j$.
(c) it retains the oldest undelivered value, (after *read*ing from, say B_i through port ρ_i) and keeps it ready for delivery.
(d) it *delivers* whatever value it has retained, whenever possible, to DEST, through the port $\bar{\delta}$. The next value is not delivered till it has been retrieved from the appropriate cell.
(e) The N cells are treated as a *circular queue* of length N (with the indices of the cells ordered as follows: $0<1<\ldots<N-1<0$). It is clear that the controller requires to maintain the following information: i, the index of the cell containing the oldest undelivered message and j, the index of the first available empty cell. Equivalently, since the values are stored *in order* in contiguous cells of the circular queue, instead of j, it may retain m the number of cells in MEM containing undelivered messages.

The state of BC is therefore determined by four arguments – the value $a \in D$ at the port α, which it may accept from SOURCE, the value $d \in D$ at the port $\bar{\delta}$, which it may deliver to DEST, the index i and the number m. As before $a=\perp$ if there is no value at α and similarly $d=\perp$ if there is no value at δ. Further, if n denotes the total number of messages in the memory-buffer controller system, then $m \leqq n \leqq m+2$ always holds.

For convenience and clarity, we classify the states of BC into eight different kinds, which reflect the choices of actions immediately available to BC in that state. In the sequel therefore, the states of BC are further decorated by subscripts which classify the state. In general, when MEM is neither full nor empty (i.e. $0<m<N$) the following are the four possible kinds of states that BC may be in.

$BC_{?r}$: may *accept* (?) from SOURCE or *read* (r) from MEM ($n=m$)

BC_{wr}: may *write* (w) into MEM or *read* (r) from MEM ($n=m+1$)

$BC_{?!}$: may *accept* (?) from SOURCE or *deliver* (!) to DEST ($n=m+1$)

$BC_{w!}$: may *write* (w) into MEM or *deliver* (!) to DEST ($n=m+2$).

Figure 4a shows the transitions that BC may go through between these four states provided MEM continues to be neither empty nor full. Otherwise, for

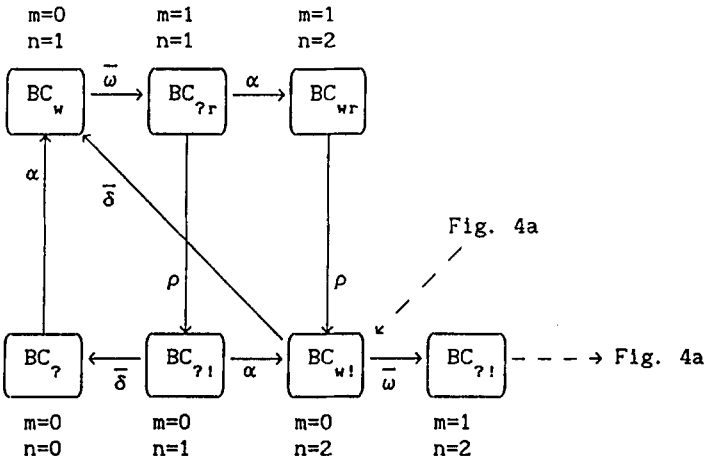


Fig. 4b. Transition diagram of BC when MEM is empty

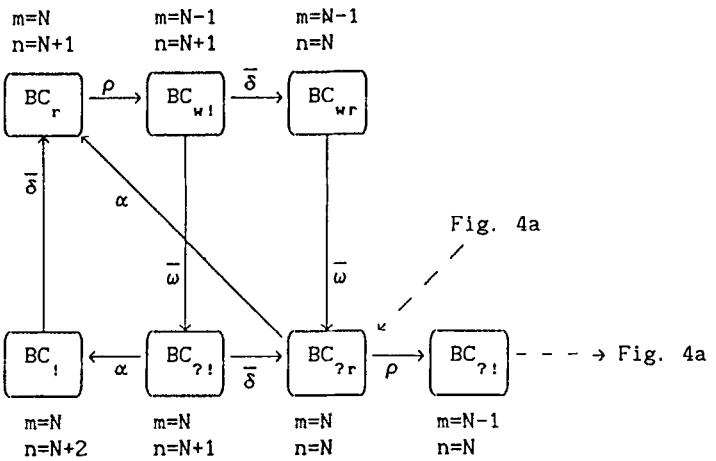


Fig. 4c. Transition diagram of BC when MEM is full

instance, when MEM is empty (i.e. $0=m\leqq n\leqq 2$) we have two more kinds of states

$BC_?$: may only *accept* (?) from SOURCE (when $n=0$) and
BW_w: may only *write* (w) into MEM (when $n=1$).

When $n=1$, depending upon the value of m (and where in the entire system the only message is), the possible kinds of states are BC_w, $BC_{?r}$, $BC_{?!}$. When $m=0$ and $n=2$, clearly the only kind of state possible is $BC_{w!}$. Similarly, when MEM is full (i.e. $N=m<n\leqq N+2$), we get two more new kinds of states, viz.

BC_r: may only *read* (r) from MEM (when $n=N+1$) and
$BC_!$: may only *deliver* (!) to DEST (when $n=N+2$).

When $N=m=n$, we have $BC_{?r}$ as the only possible kind of state. When $n=N+1$, by a reasoning similar to the case when $n=1$ (depending upon where the only "hole" or "emptiness" is) the possible kinds of states that BC could be in are BC_r, $BC_{w!}$, $BC_{?!}$. The state transition diagrams for these "boundary" conditions are shown in Figs. 4b and c respectively. We may then define $BC(a, d, i, m)$

as follows, using "$\oplus$" and "$\ominus$" to denote addition and subtraction modulo N respectively.

$$\mathrm{BC}(a,d,i,m) \equiv \begin{cases} \mathrm{BC}_{?}(a,d,i,m) & \text{if } a=\perp, d=\perp, m=0 \\ \mathrm{BC}_{w}(a,d,i,m) & \text{if } a\in D, d=\perp, m=0 \\ \mathrm{BC}_{?!}(a,d,i\,m) & \text{if } a=\perp, d\in D, 0\leqq m\leqq N \\ \mathrm{BC}_{w!}(a,d,i,m) & \text{if } a\in D, d\in D, 0\leqq m<N \\ \mathrm{BC}_{?r}(a,d,i,m) & \text{if } a=\perp, d=\perp, 0<m\leqq N \\ \mathrm{BC}_{wr}a,d,i,m) & \text{if } a\in D, d=\perp, 0<m<N \\ \mathrm{BC}_{r}(a,d,i,m) & \text{if } a\in D, d=\perp, m=N \\ \mathrm{BC}_{!}(a,d,i,m) & \text{if } a\in D, d\in D, m=N \end{cases}$$

where

$\mathrm{BC}_{?}(\perp,\perp,i,0)=\alpha(a).\mathrm{BC}_{w}(a,\perp,i,0)$

$\mathrm{BC}_{w}(a,\perp,i,0)\equiv\bar{\omega}_i(a).\mathrm{BC}_{?r}(\perp,\perp,i,1)$

$$\mathrm{BC}_{?!}(\perp,d,i,m)\equiv\begin{cases}\alpha(a).\mathrm{BC}_{w!}(a,d,i,m)+\bar{\delta}(d).\mathrm{BC}_{?}(\perp,\perp,i,m) & \text{if } m=0\\ \alpha(a).\mathrm{BC}_{w!}(a,d,i,m)+\bar{\delta}(d).\mathrm{BC}_{?r}(\perp,\perp,i,m) & \text{if } 0<m<N\\ \alpha(a).\mathrm{BC}_{!}(a,d,i,m)+\bar{\delta}(d).\mathrm{BC}_{?r}(\perp,\perp,i,m) & \text{if } m=N\end{cases}$$

$$\mathrm{BC}_{w!}(a,d,i,m)\equiv\begin{cases}\bar{\omega}_{i\oplus m}(a).\mathrm{BC}_{?!}(\perp,d,i,m+1)+\bar{\delta}(d).\mathrm{BC}_{w}(a,\perp,i,m) & \text{if } m=0\\ \bar{\omega}_{i\oplus m}(a).\mathrm{BC}_{?!}(\perp,d,i,m+1)+\bar{\delta}(d).\mathrm{BC}_{wr}(a,\perp,i,m) & \text{if } 0<m<N\end{cases}$$

$$\mathrm{BC}_{?r}(\perp,\perp,i,m)\equiv\begin{cases}\alpha(a).\mathrm{BC}_{wr}(a,\perp,i,m)+\rho_i(d).\mathrm{BC}_{?!}(\perp,d,i\oplus 1,m-1) & \text{if } 0<m<N\\ \alpha(a).\mathrm{BC}_{r}(a,\perp,i,m)+\rho_i(d).\mathrm{BC}_{?!}(\perp,d,i\oplus 1,m-1) & \text{if } m=N\end{cases}$$

$\mathrm{BC}_{wr}(a,\perp,i,m)\equiv\bar{\omega}_{i\oplus m}(a).\mathrm{BC}_{?r}(\perp,\perp,i,m+1)+\rho_i(d).\mathrm{BC}_{w!}(a,d,i\oplus 1,m-1)$

$\mathrm{BC}_{r}(a,\perp,i,N)\equiv\rho_i(d)\cdot\mathrm{BC}_{w!}(a,d,i\oplus 1,N-1)$

$\mathrm{BC}_{!}(a,d,i,N)\equiv\bar{\delta}(d)\cdot\mathrm{BC}_{r}(a,\perp,i,N).$

The reader may have realised from the above design of BC that the subscripts on the states of BC are unnecessary since its state (given by the values of the four arguments) completely determines is immediate choices. However we shall continue to use these subscripts wherever convenient, as they help in understanding the functioning of both the controller and the system as a whole. Before composing BC with MEM to obtain the complete system, we define a function

$$g: D_{\perp}^{N}\times\{0,\ldots,N-1\}\times\{0,\ldots,N\}\to D_N$$

such that

$$g(s,i,m)=\begin{cases}\varepsilon & \text{if } m=0\\ x_i & \text{if } m=1\\ x_i\,x_{i\oplus 1}\ldots x_{i\oplus m\ominus 1} & \text{otherwise,}\end{cases}$$

where x_j is the value stored in cell B_j, for all $i\leqq j<i\oplus m$. The function g abstracts away the details of *where* in MEM messages are located and yields merely the sequence, in the order of their arrivals, of undelivered messages stored in

MEM. That is, for any given $r \in D_N$ and $m = |r|$, $g(s, i, m) = r$ may hold for several different values of s, and i. Let

$$\mathrm{BUFF}_*(s, a, d, i, m) \equiv (\mathrm{MEM}(s) | \mathrm{BC}_*(a, d, i, m)) \setminus \{\rho_k, \omega_k | 0 \leqq k < N\}$$

where $* \in \{?, !, w, r, ?!, w!, ?r, wr\}$ as appropriate to the states of MEM and BC. It is easy to show that for all s, a, d, i, m and appropriately chosen values of s', a', i', m' (which may be determined from the definition), BUFF may undergo the following transitions.

$$\mathrm{BUFF}(s, \perp, d, i, m) \xrightarrow{\alpha(a)} \mathrm{BUFF}(s, a, d, i, m) \quad \text{if } a \in D, d \in D_\perp$$

$$\mathrm{BUFF}(s, a, d, i, m) \xrightarrow{\bar{\delta}(d)} \mathrm{BUFF}(s, a, \perp, i, m) \quad \text{if } a \in D_\perp, d \in D$$

$$\mathrm{BUFF}(s, a, d, i, m) \xrightarrow{1} \mathrm{BUFF}(s', a', d', i', m') \quad \text{otherwise.}$$

The last transition given above holds because the ports $\rho_k, \bar{\rho}_k, \omega_k, \bar{\omega}_k$, for all k, $0 \leqq k < N$, are hidden. The hiding operation (by internalising) the various read and write operations between MEM and BC) too has the effect of abstracting away the details of storage of the undelivered messages in MEM. We now have the following lemma.

Lemma 4.2 *1. Let* $s = x_0 \ldots x_{N-1}$ *and* $t = y_0 \ldots y_{N-1}$, *then* $g(s, i, m) = g(t, j, m)$ *iff for all* k, $0 \leqq k < m$, $x_{i \oplus k} = y_{j \oplus k}$.
2. For $s, s' \in D_\perp^N$, $i, i' \in \{0, \ldots, N-1\}$ *and* $m \in \{0, \ldots, N\}$ *if* $g(s, i, m) = g(s', i', m)$ *then* $\mathrm{BUFF}_*(s, a, d, i, m) \simeq {}^+\mathrm{BUFF}_*(s', a, d, i', m)$. *In fact they are strong congruent.*

Proof outline. 1. Follows from the fact that $g(s, i, m) = x_i \ldots x_{i \oplus m \ominus 1}$ and $g(t, j, m) = y_j \ldots y_{j \oplus m \ominus 1}$.

2. Since the hiding operation $\setminus \{\rho_k, \omega_k | 0 \leqq k < N\}$ plays essentially the same role in the overall system as the function g (in abstracting away the details of storage), it is easy to show, using the transitions of BUFF and the result in part 1 above, that the relation R

$$R = \{\langle \mathrm{BUFF}_*(s, a, d, i, m), \mathrm{BUFF}_*(s', a, d, i', m) \rangle | g(s, i, m) = g(s', i', m)\}$$

is a strong bisimulation. □

As a consequence of Lemma 4.2 we may abbreviate the state of BUFF to

$$\mathrm{BUFF}_*(dra) \equiv \mathrm{BUFF}_*(s, a, d, i, m),$$

where $r = g(s, i, m) \in D_N$. Further, since the value of "*" determines whether the ports α and/or $\bar{\delta}$ are empty or not, we may extend the function f to $\hat{f}$: $(D_\perp)_{N+2} \to D_{N+2}$, which removes all occurrences of "$\perp$" from any string of length at most $N+2$. This means for $r \in D_N$ and $a, d \in D_\perp$, we have

$$\hat{f}(dra) = \begin{cases} r & \text{if } d = a = \perp \\ dr & \text{if } d \in D, a = \perp \\ ra & \text{if } d = \perp, a \in D \\ dra & \text{if } d, a \in D. \end{cases}$$

Letting $t = \hat{f}(dra)$ it is easy to see that the values of * and t together completely specify the state of BUFF up to strong congruence. Hence we may further simplify our definition of BUFF to

$$\mathrm{BUFF}_*(t) \equiv \mathrm{BUFF}_*(dra),$$

where $\hat{f}(dra) = t \in D_{N+2}$.

Proposition 4.3 *Let* $S = \{\langle \mathrm{BUFF}_*(t), \mathrm{FIFO}(t)\rangle \mid t \in D_{N+2}\}$.
1. S is an efficiency prebisimulation.
2. $\mathrm{BUFF}_*(\varepsilon) \lesssim^+ \mathrm{FIFO}(\varepsilon)$.

Proof outline. 1. Follows from the transitions of FIFO and BUFF.

2. When $t = \varepsilon$, $* = ?$ $\mathrm{BUFF}_*(t)$ has no internal transitions. In fact, both $\mathrm{BUFF}(\varepsilon)$ and $\mathrm{FIFO}(\varepsilon)$ have only an α-transition, satisfying the conditions EP^+ of Proposition 3.11.3. □

Proposition 4.4 *Let* $T = \{\langle \mathrm{PIPE}(dsa), \mathrm{BUFF}(dra)\rangle \mid \hat{f}(s) = r, a, d \in D_\perp\}$
1. T is an efficiency prebisimulation for all $N \geqq 1$.
2. For $N = 1$ *and* $s, r \in D_\perp$ *such that* $\hat{f}(s) = r$, $\mathrm{PIPE}(dsa) \simeq^+ \mathrm{BUFF}(dra)$ *for all* $a, d \in D_\perp$.
3. For $N > 1$ *and* $s, r \in (D_\perp)_N$ *such that* $\hat{f}(s) = r$, $\mathrm{PIPE}(dsa) \lesssim^+ \mathrm{BUFF}(dra)$ *for all* $a, d \in D_\perp$. *In particular,* $\mathrm{PIPE}(\perp^{N+2}) \lesssim^+ \mathrm{BUFF}_?(\varepsilon)$.

Proof outline. 1. Similar to the proof of Proposition 4.3.1. The condition $N \geqq 1$ is absolutely necessary because if $N = 0$ then BUFF is not an implementation at all (there is no MEM)!
2. It may be seen that when $N = 1$, the relation T defined in part 1 is actually a strong bisimulation.
3. It suffices to consider only the internal transitions of the two implementations. Intuitively speaking, for every message that is accepted, PIPE has to perform $N+1$ internal actions (i.e. passing the message from cell C_{N+1} to C_N and so on till it reaches C_0) before it can deliver it to DEST. On the other hand, BUFF needs to perform exactly two internal actions (one to store the message in MEM and the other to retrieve it) before delivering it to DEST. Consider any ordered pair $\langle \mathrm{PIPE}(dsa), \mathrm{BUFF}(dra)\rangle$ in the ep T given in part 1 and let $N > 1$. The following claims may then be proved easily.

Claim 1 Let p be an integer such that $0 \leqq p \leqq N+1$, and $q = \min(2, p)$. Then for every $\langle \mathrm{PIPE}(dsa), \mathrm{BUFF}(dra)\rangle$ in T, $\mathrm{PIPE}(dsa) \xrightarrow{1^p} \mathrm{PIPE}(d'\,s'\,a')$ implies there exists $r' \in D_N$ such that $\mathrm{BUFF}(dra) \xrightarrow{1^q} \mathrm{BUFF}(d'\,r'\,a')$ and $\langle \mathrm{PIPE}(d'\,s'\,a'), \mathrm{BUFF}(d'\,r'\,a')\rangle \in T$.

Claim 2 If $\mathrm{BUFF}(dra) \xrightarrow{1} \mathrm{BUFF}(d'\,r'\,a')$ then there exists $p \geqq 1$ and s' such that $\mathrm{PIPE}(dsa) \xrightarrow{1^p} \mathrm{PIPE}(d'\,s'\,a')$ and $\langle \mathrm{PIPE}(d'\,s'\,a'), \mathrm{BUFF}(d'\,r'\,a')\rangle \in T$.

$\mathrm{PIPE}(\perp^{N+2}) \lesssim^+ \mathrm{BUFF}(\varepsilon)$ follows from the fact that both processes have no internal transitions and both may only perform α-transitions. □

5. Proof system for finite processes

In this section, we give a sound and complete (in)equational proof system for finite processes, that is, processes in which there is no occurrence of recursion.

In any proof system for finite processes in CCS, it turns out that for a precongruence which is coarser than strong bisimulation congruence ($\sim$) it is enough to restrict attention to what are referred to by Milner as "finite serial processes", i.e. processes which are built up only from prefix and summation operations. This is because, any finite process containing one or more of the so called "static operators", viz. composition, hiding and relabelling, may be transformed into a finite serial process using equations which are sound for strong bisimulation congruence. The reader may verify from the following proposition (stated without proof) that this is indeed true. It is easy to show that "|" is both commutative and associative, hence we use the unary operation "Γ" over a set of processes to denote their composition. For similar reasons "Σ" is used for summation.

Proposition 5.1 [10] *1. The Expansion Law. Let* $p \equiv \Gamma P'$, *where* $P' \equiv \{p_i \mid 1 \leqq i \leqq n\}$.

Then $p \sim \Sigma\{a_i.p' \mid p_i \xrightarrow{a_i} p'_i,\ p' \equiv \Gamma((P' - \{p_i\}) \cup \{p'_i\})\} + \Sigma\{1.p'' \mid p_i \xrightarrow{\alpha_i} p'_i,\ p_j \xrightarrow{\bar{\alpha}_i} p'_j,$
$i<j,\ p'' \equiv \Gamma((P' - \{p_i, p_j\}) \cup \{p'_i, p'_j\})\}$

2. $(a.p)\backslash L \sim \begin{cases} \mathbf{0} \text{ if } a \in L \cup \bar{L} \\ a.(p\backslash L) \text{ otherwise} \end{cases}$

3. $(a.p)[h] \sim h(a).p[h]$

4. $(p+q)\backslash L \sim (p\backslash L) + (q\backslash L)$

5. $(p+q)[h] \sim p[h] + q[h]$ □

The five parts of the above proposition may be combined and generalized into a single monolithic expansion law. However, we have given it in the above form for the sake of clarity.

A proof system for efficiency precongruence, strictly speaking, should also incorporate Proposition 5.1 as an infinitary equation (since A is an infinite set). However, it is enough for our purposes to know that such an equation exists. Our main concern therefore, will be to obtain a proof system for finite serial processes that characterizes efficiency precongruence. Our proof system, denoted $\mathcal{A}$, is shown below.

A1.	$x+y=y+x$
A2.	$x+(y+z)=(x+y)+z$
A3.	$x+x=x$
A4.	$x+\mathbf{0}=x$
A5.	$a.(x+1.x) \sqsubseteq a.x$
A6.	$1.x \sqsubseteq x+1.x$
A7.	$a.(x+1.y) \sqsubseteq a.(x+1.y)+a.y$
R0.	$\dfrac{x \sqsubseteq x+y+z}{x \sqsubseteq x+z}$

As with any inequational axiom system, provability is assumed to be closed under the following rules. Further, each of the equations A1–4 given above, actually denotes a pair of inequations which may be obtained by applying rule R2 below.

R1. $\dfrac{}{x \sqsubseteq x}$ Reflexivity

R2. $\dfrac{x \sqsubseteq y, y \sqsubseteq x}{x = y}, \dfrac{x = y}{x \sqsubseteq y, y \sqsubseteq x}, \dfrac{x = y}{y = x}$ Equality

R3. $\dfrac{x \sqsubseteq y, y \sqsubseteq z}{x \sqsubseteq z}$ Transitivity

R4. $\dfrac{x_1 \sqsubseteq y_1, \ldots, x_k \sqsubseteq y_k}{o(x_1, \ldots, x_k) \sqsubseteq o(y_1, \ldots, y_k)}$ Substitutivity

for every k-ary operator o

R5. $\dfrac{x \sqsubseteq y}{x\rho \sqsubseteq y\rho}$ for every substitution ρ Instantiation

The proof system $\mathscr{A}$ consists of the axioms A1–7 and rules R0–5. The axioms A1–4 and rules R1–5 are the usual ones an require no explanation. A5–7 are asymmetric because of the differing numbers of silent actions on the two sides of the respective inequations, i.e. their converses do not hold. Note that using A6 a slightly weaker version of A5 may be derived, $a.1.x \sqsubseteq a.x$. However, rule R0 may require some detailed intuitive explanation to justify its introduction. Let $\mathscr{A}^-$ denote the proof system $\mathscr{A}$ without the rule R0.

To explain the need for R0, it is necessary to be able to compare the proof system $\mathscr{A}^-$ with the system $\mathscr{A}'$ for observational congruence. $\mathscr{A}'$ consists of the equations A1–4, and the following equations (A5′–7′) in lieu of A5–7.

A5′. $a.1.x = a.x$

A6′. $1.x = x + 1.x$

A7′. $a.(x + 1.y) = a.(x + 1.y) + a.y$

In place of A5′ we could have used $a.(x + 1.x) = a.x$, but in view of A6′ our choice seems more straightforward. Moreover these are precisely the axioms used in [7]. Further we also have the rules R1′–5′, which are obtained by replacing all occurrences of "$\sqsubseteq$" in R1–5 by "$=$". $\mathscr{A}'$ has been shown to be complete for finite processes (see [7] or [11]). Now consider the following equation.

T1′. $z + 1.(x + y) = z + 1.(x + y) + y$

It may be easily shown that T1′ is provable in $\mathscr{A}'$ using the equation

A8′. $1.(x + y) = 1.(x + y) + y.$

A8′ in turn may be derived from A6′ in the following manner.

$$\begin{aligned} & 1.(x+y) \\ &= 1.(x+y)+(x+y) \quad \text{by A6}' \\ &= (1.(x+y)+(x+y))+y \quad \text{by A3 and A2} \\ &= 1.(x+y)+y \quad \text{by A6}' \end{aligned}$$

However, to prove the analogue of T1′, viz.

T1. $$z+1.(x+y) \sqsubseteq z+1.(x+y)+y$$

in $\mathscr{A}^-$, it would be necessary to introduce

A8. $$1.(x+y) \sqsubseteq 1.(x+y)+y$$

as an extra axiom, since A8 cannot be derived from A6 in a similar fashion. Even if A8 were added to the system $\mathscr{A}$, it is not possible to prove

T2. $$1.(x_1+1.(x_2+y)) \sqsubseteq 1.(x_1+1.(x_2+y))+y,$$

whereas, the analogue of T2, viz. T2′ is provable in $\mathscr{A}'$, without the addition of any more axioms or inference rules.

Let X denote the sequence $\langle x_i | 1 \leqq i \leqq m \rangle$ and for any y, let $a \mathbin{>\!\!-} X \square y$ denote $a.(x_1+a.(x_2+a.(\ldots+a.(x_m+y)\ldots)))$. In general, in the system $\mathscr{A}^-$, it does not seem possible to derive valid inequalities of the form,

T3. $$1 \mathbin{>\!\!-} X \square y \sqsubseteq (1 \mathbin{>\!\!-} X \square y)+y$$

without adding an infinite number of axioms starting with A8. On the other hand, by using RO in conjunction with A6, we would be able to prove inequations like T3. For instance, T2 may be derived in $\mathscr{A}$ as follows:

$$\begin{aligned} & 1.(x_1+1.(x_2+y)) \\ & \sqsubseteq 1.(x_1+1.(x_2+y))+x_1+1.(x_2+y) && \text{by A6} \\ & \sqsubseteq 1.(x_1+1.(x_2+y))+x_1+(1.(x_2+y)+x_2+y) && \text{by A6} \\ & \sqsubseteq 1.(x_1+1.(x_2+y))+(x_1+1.(x_2+y)+x_2)+y && \text{by A2} \\ & \sqsubseteq 1.(x_1+1.(x_2+y))+y && \text{by R0} \end{aligned}$$

A6 therefore enables us to extract y from an expression of the form given in T3, in which y may be nested at an arbitrary depth. However R0 enables us to discard certain other superfluous terms which also come up by the application of A6. A similar phenomenon occurs with A7. For example,

$$a.(x_1+1.(x_2+1.y)) \sqsubseteq a.(x_1+1.(x_2+1.y))+a.y$$

cannot be derived in $\mathscr{A}$ although the corresponding identity is a theorem in $\mathscr{A}'$.

We give below the proof of soundness of R0 and leave the other proofs of soundness to the reader. We then proceed to show that the proof system $\mathscr{A}$ is complete for finite processes.

Proposition 5.2 *Rule R0 is sound.*

Proof. Suppose $p \lesssim^+ p+q+r$. Let $p_1 \equiv p+q+r$ and $p_2 \equiv p+r$. By Proposition 3.11.3 we have

(1) $$p \xrightarrow{a} p' \Rightarrow \exists p'_1 : p_1 \xrightarrow{a} p'_1 \wedge p' \lesssim p'_1$$

(2) $$p_1 \xrightarrow{a} p'_1 \Rightarrow \exists p' : p \stackrel{a}{\Rightarrow} p' \wedge p' \lesssim p'_1$$

Letting $p'_2 \equiv p'$ we have

(3) $$p \xrightarrow{a} p' \Rightarrow \exists p'_2 : p_2 \xrightarrow{a} p'_2 \wedge p' \lesssim p'_2$$

From (2) it follows that

(4) $$p_2 \xrightarrow{a} p'_2 \Rightarrow \exists p' : p \stackrel{a}{\Rightarrow} p' \wedge p' \lesssim p'_2$$

By (3), (4) and Proposition 3.11.3 we get $p \lesssim^+ p+r$. □

Lemma 5.3 *For all processes p, q, $p \lesssim p$ implies $p \lesssim^+ q$ or $p \lesssim^+ q+1.q$.*

Proof. Assume $p \lesssim q$ and $p \lesssim^+ q$. We have to show $p \lesssim^+ q+1.q$. By Proposition 3.11 it is sufficient to show

(1) $$p \xrightarrow{a} p' \Rightarrow \exists q' : q+1.q \xrightarrow{a} q' \wedge p' \lesssim q'$$

(2) $$q+1.q \xrightarrow{a} q' \Rightarrow \exists p' : p \stackrel{a}{\Rightarrow} p' \wedge p' \lesssim q'$$

(1) follows from the fact that $p \lesssim q$ using the characterization of $\lesssim$ given in Proposition 3.6. The same characterization ensures (2) whenever $q \xrightarrow{a} q'$. Hence it remains to prove $\exists p' : p \stackrel{1}{\Rightarrow} p' \wedge p' \lesssim q$ when $q+1.q \xrightarrow{1} q$. Since $p \lesssim q$ and $p \not\lesssim^+ q$, from EP′2 and the negation of EP^+2 for $a=1$, we obtain

$$p \xrightarrow{1} p' \wedge p' \lesssim q \wedge (\forall q' : q \xrightarrow{1} q' : p' \not\lesssim q').$$

Hence

$$q+1 \cdot q \xrightarrow{1} q \Rightarrow \exists p' : p \stackrel{1}{\Rightarrow} p' \wedge p' \lesssim q$$

which was required. □

Lemma 5.4 $p \xrightarrow{a} p'$ *implies* $\mathscr{A} \vdash p = p + a.p'$.

Proof. By induction on the depth of inference of $p \xrightarrow{a} p'$. □

Lemma 5.5 *$p \overset{a}{\Rightarrow} p'$ implies $\mathscr{A} \vdash p \sqsubseteq p + a.p'$.*

Proof. By induction on the number of silent actions involved in the transition $p \overset{a}{\Rightarrow} p'$. If $p \overset{a}{\Rightarrow} p'$ then the result follows from Lemma 5.4. Otherwise we have two cases to consider. Either $p \xrightarrow{1} q \overset{a}{\Rightarrow} p'$ or $p \overset{a}{\Rightarrow} q \xrightarrow{1} p'$. In the former case by Lemma 5.4 we have

(1)	$\mathscr{A} \vdash p = p + 1.q$	
	$\mathscr{A} \vdash q \sqsubseteq q + a.p'$	by induction hypothesis
	$\mathscr{A} \vdash 1.q \sqsubseteq 1.(q + a.p')$	by R4
(2)	$\mathscr{A} \vdash 1.q \sqsubseteq 1.(q + a.p') + q + a.p'$	by A6
(3)	$\mathscr{A} \vdash p \sqsubseteq p + (1.(q + a.p') + q) + a.p'$	by (1), (2), R3 and A2

Applying rule R0 to (3) we obtain

$$\mathscr{A} \vdash p \sqsubseteq p + a.p'.$$

In the latter case by the induction hypothesis

(4)	$\mathscr{A} \vdash p \sqsubseteq p + a.q$	
	$\mathscr{A} \vdash q = q + 1.p'$	by Lemma 5.4
	$\mathscr{A} \vdash a.q = a.(q + 1.p')$	by R4
(5)	$\mathscr{A} \vdash a.q \sqsubseteq a(q + 1.p') + a.q$	by A7
(6)	$\mathscr{A} \vdash p \sqsubseteq p + a.(q + 1.p') + a.q$	from (4), (5) and R3
	$\mathscr{A} \vdash p \sqsubseteq p + a.p'$	from (6) and R0. □

Theorem 5.6 *(Completeness). $p \lesssim^+ q$ implies $\mathscr{A} \vdash p \sqsubseteq q$*

Proof. We prove this by induction on the sum of the *depths* of p and q, where the *depth* of p is the maximum number of nested prefixes in p. We may assume p and q have the standard forms $\Sigma\{a_i.p_i \mid 1 \leqq i \leqq m\}$ and $\Sigma\{b_j.q_j \mid 1 \leqq j \leqq n\}$ respectively (and for all i, j, $1 \leqq i \leqq m$, $1 \leqq j \leqq n$, p_i, q_j are in standard form). If the sum of the depths is 0 (i.e. $m = n = 0$), there is nothing to prove since $p \equiv \mathbf{0} \equiv q$. Otherwise, by repeated applications of A4, all summands that are $\mathbf{0}$ may be eliminated leaving p and q in standard form ($m, n > 0$). We prove the following claim.

Claim. (i) $\mathscr{A} \vdash a_i.p_i + q \sqsubseteq q$ for every i, $1 \leqq i \leqq m$.
(ii) $\mathscr{A} \vdash p \sqsubseteq p + b_j.q_j$ for every j, $1 \leqq j \leqq n$.

(i) If $p \lesssim^+ q$ then since $p \xrightarrow{a_i} p_i$ there exists $b_j = a_i$ with $q \xrightarrow{bj} q_j$ and $p_i \lesssim q_j$. By Lemma 5.3 $p_i \lesssim^+ q_j$ or $p_i \lesssim^+ q_j + 1.q_j$. By the induction hypothesis we have $\mathscr{A} \vdash p_i \sqsubseteq q_j$ or $\mathscr{A} \vdash p_i \sqsubseteq q + 1.q_j$. In the former case $\mathscr{A} \vdash a_i.p_i \sqsubseteq b_j.q_j$ by R4, and

in the latter we get the same by applying R4 followed by A5. The result then follows.

(ii) If $q \xrightarrow{b_j} q_j$ then by Proposition 3.11.3 there exists p' and some $a_i = b_j$ such that $p \overset{a_i}{\Rightarrow} p'$ and $p' \lesssim q_j$. By Lemma 5.3 and the induction hypothesis we get $\mathscr{A} \vdash p' \sqsubseteq q_j$ or $\mathscr{A} \vdash p' \sqsubseteq q_j + 1.q_j$. It then follows (possibly by applying A5) that $\mathscr{A} \vdash b_j.p' \sqsubseteq b_j.q_j$ and by Lemma 5.5 $\mathscr{A} \vdash p \sqsubseteq p + b_j.p'$ from which we obtain $\mathscr{A} \vdash p \sqsubseteq p + b_j.q_j$.

Having proved the claim, we proceed as follows. By R4, we may sum up (i) for all i, reorder the terms (by A1 and A2) and repeatedly apply A3, to obtain $\mathscr{A} \vdash p + q \sqsubseteq q$. By performing similar operations with (ii) for all j, we get $\mathscr{A} \vdash p \sqsubseteq p + q$. The result then follows by R3. □

6. Conclusion

The efficiency preorder we have introduced, $\lesssim$, is based on the simple idea of, essentially, counting the number of internal moves made by a process. We have shown that this idea may be successfully incorporated within the general framework of bisimulations, [11], to obtain a mathematically tractable preorder, which, in common with the standard notions of bisimulation equivalence, is sensitive to the branching structure of processes yet supports abstraction. It is mathematically tractable in that it is preserved by all CCS contexts and we have given a complete proof system for finite terms, based on a modification of the standard τ-laws for CCS. Moreover the usual algorithms for checking bisimulations or finding bisimulations, [3], may easily be adapted to $\lesssim$. It supports abstraction in that it is insensitive to many of the internal details of processes. For example, the usual laws associated with port restrictions are true as are variations on the τ-laws. It differs from the abstraction supported by weak bisimulation only in that processes may be differentiated if their response times to external stimuli are different. This is also the basis of comparing and ordering processes: $p \lesssim q$ if the response time of q to external stimuli is uniformly faster than that of p.

Although the basis of the preorder is very simple, we have demonstrated its usefulness by applying it to an example of the implementation of a FIFO queue. In future work we hope to apply it to more significant examples. We also intend to investigate the possibility of characterising $\lesssim$ completely using a finite set of inequations. Recall that the proof rule R0 is essential to our proof system.

There has been much recent work on introducing notions of time into process algebras. [2] and [14] are typical examples of one approach, where real-time durations are associated with actions. [12] and [8] are examples of another approach where actions are still instantaneous but a special action is introduced to represent the passage of time. Neither of these approaches is directly comparable with the one presented here, which we believe to be the first "improvement" preorder based on time.

References

1. Arun-Kumar, S., Hennessy, M.: An efficiency preorder for processes. Proc. International Conference on Theoretical Aspects of Computer Software. (Lect. Notes Comput. Sci., vol. 526, pp. 152–175) Berlin Heidelberg New York: Springer 1991
2. Beaton, J., Bergstra, J.: Real time process algebra, Technical Report CWI Amsterdam, 1989
3. Cleaveland, R., Parrow, J., Steffen, B.: The concurrency workbench: A semantics-based verification tool for finite-state systems, Technical Report ECS-LFCS-89-83, University of Edinburgh, 1989
4. Davies, J., Schneider, S.: An introduction to timed CSP, Technical Report, PRG, Oxford, 1989
5. Gerth, R., Boucher, A.: A timed failures model for extended communicating sequential processes (Lect. Notes Comput. Sci., vol. 267, pp. 95–114) Berlin Heidelberg New York: Springer 1986
6. Hennessy, M.: Algebraic theory of processes. Cambridge, MA: MIT Press 1988
7. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM **32**(1), 137–161 (1985)
8. Hennessy, M., Regan, T.: A temporal process algebra. Technical Report 2/90, University of Sussex, 1990
9. C.A.R. Hoare: Communicating sequential processes. Englewood Cliffs, NJ: Prentice Hall 1985
10. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. **25**, 267–310 (1983)
11. Milner, R.: Communicationa and concurrency. Englewood Cliffs, NJ: Prentice Hall 1989
12. Nicollin, X., Richier, J.L., Sifakis, J., Voiron, J.: ATP: An algebra for timed processes. Technical Report, Grenoble, 1989
13. Park, D.: Concurrency and automata on infinite sequences. (Lect. Notes Comput. Sci., vol. 104, pp. 167–183) Berlin Heidelberg New York: Springer 1980
14. Reed, G.M., Roscoe, A.: A timed model for communicating sequential processes (Lect. Notes Comput. Sci., vol. 226, pp. 314–323) Berlin Heidelberg New York: Springer 1986
15. Rudkin, S., Smith, C.R.: A temporal enhancement for LOTOS. British Telecom, 1988