An Efficiency Preorder for Processes
(Revised, corrected and extended)

S. Arun-Kumar*
Department of Computer Science and Engineering,
Indian Institute of Technology,
Hauz Khas, New Delhi 110 016 INDIA.
Email: sak@cse.iitd.ernet.in

M. Hennessy†
School of Computing and Cognitive Sciences
University of Sussex
Falmer, Brighton BN1 9QH, ENGLAND.
Email: matthewh@cogs.sussex.ac.uk

October 28, 1998

Abstract

A simple efficiency preorder for CCS processes is introduced, in which $p \precsim q$ 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: different implementations of a bounded buffer. Finally a sound and complete proof system for finite processes is given.

This paper appeared earlier in [1] and [2]. This version of the paper has been revised, corrected and extended to include more operators, more examples and some additional remarks and observations by the first author. Hence all errors are entirely his responsiblity.

*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 ESPIRIT II.

1 Introduction

A large number of behavioural equivalences for process description languages have been studied in recent years ([10], [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 depends at least to some extent on the fine-tuning of 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" [13] and timed CSP [12]. 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 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: $p \precsim q$ 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 $\alpha.1.\beta.\mathbf{0} \precsim \alpha.\beta.\mathbf{0}$ but $\alpha.\beta.\mathbf{0} \not\precsim \alpha.1.\beta.\mathbf{0}$.

One obvious advantage of such a treatment as opposed to the usual real-time equivalences such as [12] 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 applicable to more general languages with real-time features.

In section 2 we define the syntax and operational semantics of CCS. This is completely standard and may be omitted by readers familiar with CCS. In section 3 we show how to modify the usual recursive definition of bisimulation equivalence [10], so as to obtain $\precsim$. Roughly speaking for $p \precsim q$ to hold, every "weak arrow", $\overset{a}{\Longrightarrow}$, 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 [10]) it is not preserved by the choice 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 recursion. We also justify fixpoint induction. In section 5, we consider an example which examines different methods of implementing a FIFO buffer. In section 4 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 Λ and $\bar{\Lambda}$ (the complement of Λ) be infinite disjoint sets in bijection under the complementation operation "$\bar{\ }$". Then $V = \Lambda \cup \bar{\Lambda}$ 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 complementation 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 lower case greek letters (possibly decorated) denote visible actions (that is, elements of $V \times D$). Initial lower case roman letters ($a, b, c, \ldots$ suitably decorated) range over (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.

$$e ::= x \mid \mathbf{0} \mid \alpha(u).e \mid \bar{\alpha}(d).e \mid 1.e \mid \sum \tilde{e} \mid e \mid e \mid e \rhd e \mid e \backslash L \mid e[h] \mid \mu_i \vec{x} : \vec{e}$$

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

We will not explicitly specify elements of the domain or value variable names unless required in the context. 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 A. Any term generated by the

$$
\begin{array}{llcl}
(P1) & & & \alpha(u).e \xrightarrow{\alpha(d)} e\{d/u\}, \quad \text{for all } d \in D \\
(P2) & & & \bar{\alpha}(d).e \xrightarrow{\bar{\alpha}(d)} e \\
(P3) & & & 1.e \xrightarrow{1} e \\
(S) & e_i \xrightarrow{a} e_i' & \Longrightarrow & \sum \tilde{e} \xrightarrow{a} e_i' \\
(D1) & e \xrightarrow{a} e' & \Longrightarrow & e \rhd f \xrightarrow{a} e' \rhd f \\
(D2) & f \xrightarrow{a} f' & \Longrightarrow & e \rhd f \xrightarrow{a} f' \\
(C1) & e \xrightarrow{a} e' & \Longrightarrow & e \mid f \xrightarrow{a} e' \mid f \\
(C2) & f \xrightarrow{a} f' & \Longrightarrow & e \mid f \xrightarrow{a} e \mid f' \\
(C3) & e \xrightarrow{\alpha} e',\ f \xrightarrow{\bar{\alpha}} f' & \Longrightarrow & e \mid f \xrightarrow{1} e' \mid f' \\
(R) & e \xrightarrow{a} e' & \Longrightarrow & e[h] \xrightarrow{h(a)} e'[h] \\
(H) & e \xrightarrow{a} e',\ a \notin L \cup \bar{L} & \Longrightarrow & e\backslash L \xrightarrow{a} e'\backslash L \\
(\mu) & e_i\{\mu\vec{x} : \vec{e}/\vec{x}\} \xrightarrow{a} e' & \Longrightarrow & \mu_i\vec{x} : \vec{e} \xrightarrow{a} e'
\end{array}
$$

Figure 1: The operational semantics of CCS expressions

above BNF is called a *process expression* and $\mathcal{E}$, 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) and $p, q, r, \ldots$ (possibly decorated) range over the set $\mathcal{P}$ 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. We have omitted them for the sake of simplicity. However, in this version we have added generalized summation ($\sum$) and the "disabling" operator ($\rhd$) from LOTOS. The operational semantics of the language is defined in terms of labelled transition systems in the usual fashion. Note that the generalized summation allows for an arbitrary number (finite or infinite) of summands (the $\tilde{e}$ denotes this). But in most examples we will use only the binary summation operator "+".

Let $\langle \mathcal{E}, A, \{\xrightarrow{a} \mid a \in A\}\rangle$ be a labelled transition system (LTS) where the transition relation $\longrightarrow\ \subseteq\ \mathcal{E} \times A \times \mathcal{E}$ is the smallest relation satisfying the axioms and rules of inference in figure 1. $f\{\vec{r}/\vec{x}\}$ denotes the syntactic substitution of all free occurrences of the variable x_j ($x_j \in \vec{x}$) in the expression f by r_j ($r_j \in \vec{r}$) for each j in the indexing set of the vectors $\vec{x}$ and $\vec{r}$. Similarly in the rule (P1) $\{d/u\}$ denotes the binding of the value variable u to the value $d \in D$.

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* $\langle \mathcal{E}, A^*, \{\xrightarrow{s} \mid s \in A^*\}\rangle$, *where* $\longrightarrow\subseteq \mathcal{E} \times A^* \times \mathcal{E}$ *is the least relation satisfying the following conditions:*

- $p \xrightarrow{\varepsilon} p$ *for all* p.
- $p \xrightarrow{s} p'$ *for* $s = at$ *iff* $\exists p'' : p \xrightarrow{a} p'' \xrightarrow{t} p'$.

Definition 2.2 *A* **weak LTS** *is a structure* $\langle \mathcal{E}, V^*, \{\stackrel{s}{\Longrightarrow} \mid s \in V^*\}\rangle$, *where for* $s = \alpha_1 \ldots \alpha_n \in V^*$, $e \stackrel{s}{\Longrightarrow} e'$ *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* $m_1, \ldots, m_n \geq 0$ *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 $p \Longrightarrow p'$ to denote $p(\xrightarrow{1})^* p'$ and $p \stackrel{1}{\Longrightarrow} p'$ to denote $p(\xrightarrow{1})^+ p'$.

Definition 2.3 *Let* $\leq$ *be the binary relation on* A *generated by the inequation* $1s \leq s$ *i.e.* $\leq$ *is closed under reflexivity, transitivity, and substitution under catenation contexts. An* **extended action** *is an element of* A^* *containing* at most one *visible action.*

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

The following 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.

Definition 2.4 *If* $\trianglelefteq$ *is a binary relation on processes and* e *and* f *are process expressions such that* $FPV(e) \cup FPV(f) \subseteq \vec{x}$ *then* $e \trianglelefteq f$ *iff for every vector* $\vec{p}$ *of processes,* $e\{\vec{p}/\vec{x}\} \trianglelefteq f\{\vec{p}/\vec{x}\}$.

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 [10] for the relevant proofs.

3 Weak Bisimulations and Efficiency Prebisimulations

We now define the notion of weak bisimulations [10] and state some of its properties.

Definition 3.1 *A binary relation* $R \subseteq \mathcal{P} \times \mathcal{P}$ *is a* **weak bisimulation** *(abbreviated to* **wb***) if for every* $\langle p, q\rangle \in R$ *and* $t \in V^*$ *the following conditions are satisfied.*

$$p \stackrel{t}{\Longrightarrow} p' \implies \exists q' : q \stackrel{t}{\Longrightarrow} q' \wedge p'Rq' \tag{1}$$

$$q \stackrel{t}{\Longrightarrow} q' \implies \exists p' : p \stackrel{t}{\Longrightarrow} p' \wedge p'Rq' \tag{2}$$

Proposition 3.2

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

The relation $\approx$ is called **observational equivalence**.

Definition 3.3 *A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is an* **efficiency prebisimulation** *(***ep** *for short) if for every $\langle p, q\rangle \in R$ and $s, t \in EA$ the following conditions are satisfied.*

$$p \xrightarrow{s} p' \implies \exists t : s \leq t : \exists q' : q \xrightarrow{t} q' \wedge p'Rq' \tag{3}$$

$$q \xrightarrow{t} q' \implies \exists s : s \leq t : \exists p' : p \xrightarrow{s} p' \wedge p'Rq' \tag{4}$$

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 3.4

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. *$\precsim\ = \bigcup\{R \mid R$ is an ep$\}$ is the largest ep.*
5. *$p \precsim q$ iff for some ep R, pRq.*
6. *$\precsim$ is a preorder on $\mathcal{P}$.*

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

Proposition 3.5 *A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is an ep iff for every $\langle p, q\rangle \in R$, $\alpha \in V$, $a \in A$ the following conditions are satisfied.*

$$p \xrightarrow{\alpha} p' \implies \exists q' : q \xrightarrow{\alpha} q' \wedge p'Rq' \tag{5}$$

$$p \xrightarrow{1} p' \implies p'Rq \vee (\exists q' : q \xrightarrow{1} q' \wedge p'Rq') \tag{6}$$

$$q \xrightarrow{a} q' \implies \exists p' : p \stackrel{a}{\Longrightarrow} p' \wedge p'Rq' \tag{7}$$

Proof. ($\Longrightarrow$) It is easy to see that (3) implies (5) because for $s = \alpha$, $s \leq t$ implies $t = \alpha$. (3) also implies (6) since for $s = 1$, $s \leq t$ implies either $t = \varepsilon$ or $t = 1$. Also it is obvious that (4) implies (7) for $t = \alpha \in V$. For $t = 1$, from (4) we have $p \stackrel{s}{\longrightarrow} p'$ for some $s \leq t$, that is, $p \stackrel{1}{\Longrightarrow} p'$. Hence (3.3) implies (5)-(7).
($\Longleftarrow$) Suppose R is a relation satisfying the conditions (5)-(7). Let $\langle p, q\rangle \in R$ and for some $s \in EA$, let $p \stackrel{s}{\longrightarrow} p'$. We proceed by induction on the length of s. If $s = \varepsilon$ there is nothing to prove. Otherwise $s = as_1$ for some $a \in A$ and $s_1 \in EA$. There exists p_1 such that $p \stackrel{a}{\longrightarrow} p_1 \stackrel{s_1}{\longrightarrow} p'$. Now we have two cases to consider.
Case (i) $\underline{a = \alpha \in V}$. By (5) there is a q_1 such that $q \stackrel{\alpha}{\longrightarrow} q$ and $p_1 R q_1$. By the induction hypothesis there exists $t_1 \in EA$ and q' such that $s_1 \leq t_1$, $q_1 \stackrel{t_1}{\longrightarrow} q'$ and $p' R q'$. Letting $t = \alpha t_1$ it is easy to see that (3) holds.
Case (ii) $\underline{a = 1}$. Then either $p_1 R q$ or there is a q_1 such that $q \stackrel{1}{\longrightarrow} q_1$ 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 t_1, $s_1 \leq t_1$ and q' such that $q \stackrel{t_1}{\longrightarrow} q'$ and $p' R q'$. Letting $t = t_1$ we have $s = 1s_1 \leq t_1 = t$ and (3) follows.
It is obvious that (4) follows from (7). ■

As in the case of $\approx$ (see [10]) we may show that $\precsim$ is preserved by all operators except summation. As the following example shows, it is due to the preemptive power of the silent action that $\approx$ and $\precsim$ are not preserved under summation.

Example 3.6 $1.\alpha.\mathbf{0} \precsim \alpha.\mathbf{0}$ *whereas* $1.\alpha.\mathbf{0} + \beta.\mathbf{0} \not\precsim \alpha.\mathbf{0} + \beta.\mathbf{0}$ *because* $1.\alpha.\mathbf{0} + \beta.\mathbf{0} \stackrel{1}{\longrightarrow} \alpha.\mathbf{0}$ *while* $\alpha.\mathbf{0} + \beta.\mathbf{0} \stackrel{\varepsilon}{\longrightarrow} \alpha.\mathbf{0} + \beta.\mathbf{0}$. *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 $\precsim$. In the following lemma we state without proof the result that all the operators, except summation and recursion, preserve the preorder.

Lemma 3.7 *Let* $\trianglelefteq \in \{\precsim, \approx\}$. *Then for all* $p,\ q \in \mathcal{P}$, $p \trianglelefteq q$ *implies for all* $r \in \mathcal{P}$,

1. $a.p \trianglelefteq a.q$ *for all* $a \in A$
2. $p \mid r \trianglelefteq q \mid r$ *and* $r \mid p \trianglelefteq r \mid q$
3. $p\backslash L \trianglelefteq q\backslash L$ *for all* $L \subseteq V$,
4. $p[h] \trianglelefteq q[h]$ *for any relabelling function* h.
5. $p \triangleright r \trianglelefteq q \triangleright r$
6. $r \triangleright p \trianglelefteq r \triangleright q$

Proof. It suffices in each case to find an appropriate bismulation (either efficiency or weak). As an illustration we prove that $\precsim$ is preserved under the two cases of the disabling operator by constructing a single relation for both. We show that the relation $R = \precsim \cup S$, where

$$S = \{\langle p \triangleright r, q \triangleright r\rangle, \langle r \triangleright p, r \triangleright q\rangle \mid p \precsim q,\ p,\ q,\ r \in \mathcal{P}\}$$

is an ep. Consider any pair $\langle p \triangleright r, q \triangleright r \rangle \in S$. We have the following cases to consider.
Case (i). $\underline{p \triangleright r \xrightarrow{\alpha} pr'}$ for $\alpha \in V$. Then either $p \xrightarrow{\alpha} p'$ and $pr' \equiv p' \triangleright r$ or $r \xrightarrow{\alpha} r'$ and $pr' \equiv r'$. In the former instance, since $p \precsim q$, there exists q' such that $q \xrightarrow{\alpha} q'$ and $p' \precsim q'$. We then have $q \triangleright r \xrightarrow{\alpha} qr' \equiv q' \triangleright r$ and $\langle pr', qr' \rangle \in R$. If on the other hand $r \xrightarrow{\alpha} r'$, then it follows that $q \triangleright r \xrightarrow{\alpha} r' \equiv qr' \equiv pr'$ and $pr'Rqr'$.
Case (ii). $\underline{p \triangleright r \xrightarrow{1} pr'}$. Then again either $p \xrightarrow{1} p'$ and $pr' \equiv p' \triangleright r$ or $r \xrightarrow{1} r'$ and $pr' \equiv r'$. The case $r \xrightarrow{1} r'$ is handled in a manner simlar to case (i). In the former instance, if $p' \precsim q$, then we have $pr' \precsim qr' \equiv q \triangleright r$. Otherwise, for some q', $q \xrightarrow{1} q'$ and $p' \precsim q'$. It then follows that $pr'Rqr'$ where $qr' \equiv q' \triangleright r$.
Case (iii). $\underline{q \triangleright r \xrightarrow{a} qr'}$. Again if $qr' \equiv r'$ with $r \xrightarrow{a} r'$, the proof is trivial. But if $qr' \equiv q' \triangleright r$, with $q \xrightarrow{a} q'$, then since $p \precsim q$, we have for some p', $p \overset{a}{\Longrightarrow} p'$ and $p' \precsim q'$. Letting $pr' \equiv p' \triangleright r$, it follows that $p \triangleright r \overset{a}{\Longrightarrow} pr'$ and $pr'Rqr'$.

A similar proof may be formulated for an arbitrary pair $\langle r \triangleright p, r \triangleright q \rangle$ with $p \precsim q$. The proof for $\approx$ follows along similar lines. ■

It is also easy to verify that the 1-laws (called "τ-laws" in [10]) no longer hold symmetrically for the preorder $\precsim$. We state these laws in the next lemma. In the sequel we always use $\trianglelefteq$ to denote either $\precsim$ or $\approx$.

Lemma 3.8

1. $1.p \trianglelefteq p$
2. $1.p \trianglelefteq p + 1.p \trianglelefteq p$
3. $a.1.p \trianglelefteq a.p$
4. $a.(p + 1.q) \trianglelefteq a.(p + 1.q) + a.q$

This lemma may be proved for both $\precsim$ and $\approx$ by constructing appropriate eps (note that it is not enough to construct wbs for the purpose of $\precsim$, though every ep is a wb by proposition (3.4)).

Definition 3.9 *For $\trianglelefteq \in \{\precsim, \approx\}$, $p \trianglelefteq^+ q$ iff for some visible action α not occurring in p or q, $p + \alpha.\mathbf{0} \trianglelefteq^+ q + \alpha.\mathbf{0}$.*

For reasons that will become clear in theorem (3.9), the relations $\precsim^+$ and $\approx^+$ defined above are called **efficiency precongruence** and **observational congruence** respectively. In the next proposition we give a useful behavioural characterization of $\precsim^+$.

Proposition 3.10

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

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

$$p \xrightarrow{a} p' \implies \exists q' : q \xrightarrow{a} q' \wedge p' \precsim q' \tag{8}$$

$$q \xrightarrow{a} q' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p' \precsim q' \tag{9}$$

Proof.

1. Obvious.

2. Let $p \precsim^+ q$. Then by definition 3.9, $p + \alpha.\mathbf{0} \precsim\ q + \alpha.\mathbf{0}$, 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 (7), we get $p' \precsim q + \alpha.\mathbf{0}$ or for some q', $q + \alpha.\mathbf{0} \xrightarrow{1} q'$ and $p' \precsim q'$. But $p' \not\precsim q + \alpha.\mathbf{0}$, 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 $p' \precsim q'$.

3. ($\implies$) Follows from the previous parts.
   ($\impliedby$) It suffices to show from (8) and (9) that $p + \alpha.\mathbf{0} \precsim\ q + \alpha.\mathbf{0}$. It is easy to show from (8) (respectively (9)) that (5) (respectively (7)) is satisfied by the pair $\langle p + \alpha.\mathbf{0}, q + \alpha.\mathbf{0}\rangle$. Let $p + \alpha.\mathbf{0} \xrightarrow{1} p'$, then since $\alpha \in V$, we have $p \xrightarrow{1} p'$ and from (8) it follows that for some q', $q \xrightarrow{1} q'$ and $p' \precsim\ q'$. Hence $q + \alpha.\mathbf{0} \xrightarrow{1} q'$ and $p' \precsim q'$ which proves (7). ■

A similar behavioural characterization exists for $\approx^+$ which we merely reproduce without proof ([10]).

Proposition 3.11 $p \approx^+ q$ *iff for every $a \in A$, the following conditions are satisfied.*

$$p \xrightarrow{a} p' \implies \exists q' : q \overset{a}{\Longrightarrow} q' \wedge p' \approx q' \tag{10}$$

$$q \xrightarrow{a} q' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p' \approx q' \tag{11}$$

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

Proposition 3.12 $\trianglelefteq^+$ *is preserved under prefix, summation, disabling, composition, hiding and relabelling.*

Proof. We show only the proof for summation and the reader may refer to [10] for the other proofs. Assume $\tilde{p} = \{p_i \mid i \in I\}$ and $\tilde{q} = \{q_i \mid i \in I\}$ are two sets of processes such that $\tilde{p} \trianglelefteq^+ \tilde{q}$, that is $p_i \trianglelefteq^+ q_i$ for each $i \in I$. We need to show that $\sum \tilde{p} \trianglelefteq^+ \sum \tilde{q}$. Since each pair $\langle p_i, q_i \rangle$ satisfies the conditions (8) and (9) in the case of $\precsim^+$ and the conditions (10) and (11) in the case of $\approx^+$, it follows that $\sum \tilde{p} \xrightarrow{a} p'$ only if for some $i \in I$, $p_i \xrightarrow{a} p'$. This implies (in the case of $\precsim^+$), $q_i \xrightarrow{a} q'$ and $p' \precsim^+ q'$ and hence $\sum \tilde{q} \xrightarrow{a} q'$ and $p' \precsim^+ q'$. In the case of $\approx^+$ a similar argument with $\xrightarrow{a}$ replaced by $\overset{a}{\Longrightarrow}$ holds.

Again if $\sum \tilde{q} \xrightarrow{a} q'$ then for some $j \in I$, we have $q_j \xrightarrow{a} q'$ which implies for some p', $p_i \overset{a}{\Longrightarrow} p'$ and $p' \trianglelefteq^+ q'$. Hence $\sum \tilde{p} \overset{a}{\Longrightarrow} p'$ and $p' \trianglelefteq^+ q'$. ■

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

Lemma 3.13 *Let $\vec{e}$ and $\vec{f}$ be process expressions such that $\vec{e} \trianglelefteq^+ \vec{f}$ and $FPV(\vec{e}) \cup FPV(\vec{f}) \subseteq \vec{x}$. Then $\mu\vec{x} : \vec{e} \trianglelefteq^+ \mu\vec{x} : \vec{f}$.*

Proof outline. This was originally proved[1] in [10] for $\approx^+$ and the same proof may be easily adapted for $\trianglelefteq^+$. We give an outline for the case of e and f having at most one free variable, say x. That is, we prove that $e \trianglelefteq^+ f$ implies $p \trianglelefteq^+ q$ where $p \equiv \mu x : e$ and $q \equiv \mu x : f$. Now consider the relation

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

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 the following conditions hold for all $a \in A$.

1. $g\{p/x\} \xrightarrow{a} p' \implies \exists q' : g\{q/x\} \xrightarrow{a} q' \wedge p' R{\circ}{\trianglelefteq} q'$
2. $g\{q/x\} \xrightarrow{a} q' \implies \exists p' : g\{p/x\} \xrightarrow{a} q' \wedge p' {\trianglelefteq}{\circ}R q'$

From (1 and (2) we may show that ${\trianglelefteq}{\circ}R{\circ}{\trianglelefteq}$ is an ep if $\trianglelefteq = \precsim$ and a wb if $\trianglelefteq = \approx$, and since $R \subseteq {\trianglelefteq}{\circ}R{\circ}{\trianglelefteq}$ it follows that $R \subseteq \trianglelefteq$. Letting $g \equiv x$ it is clear that $\langle p, q\rangle \in R$ and hence $p \trianglelefteq q$. But from the two conditions above, conditions (8) and (9) in proposition (3.10) and conditions (10) and (11), we obtain the stronger result $p \trianglelefteq^+ q$. ■

Theorem 3.14 *$\trianglelefteq^+$ is a precongruence on CCS (with $\triangleright$) process expressions.*

Proof. Follows from proposition (3.12) and lemma (3.13). ■

Corollary 3.15

1. *$p \precsim^+ q$ iff pRq for some ep R and for all $a \in A$,*
$$p \xrightarrow{a} p' \implies \exists q' : q \xrightarrow{a} q' \wedge p'Rq' \quad \& \quad q \xrightarrow{a} q' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p'Rq'$$
2. *$p \approx^+ q$ iff pRq for some wb R and for all $a \in A$.*
$$p \xrightarrow{a} p' \implies \exists q' : q \overset{a}{\Longrightarrow} q' \wedge p'Rq' \quad \& \quad q \xrightarrow{a} q' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p'Rq'$$
3. *$p \precsim^+ q$ implies $p \approx^+ q$.*

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

We will also have occasion to refer to another relation which we briefly describe here. If definition (3.1) were strengthened by letting $t \in A^*$ and replacing "$\overset{t}{\Longrightarrow}$" everywhere by "$\xrightarrow{t}$" in the conditions (1) and (2), then the relation obtained is a **strong bisimulation** and $\sim$, called **strong congruence**, denotes the largest strong bisimulation (for details see [10]). 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 examples shown in figures 2 and 3 make clear the difference between the three relations.

[1] incorrectly, though it was later corrected in [14]. This proof for the general relation $\trianglelefteq^+$ however predates [14] and we believe this proof is correct

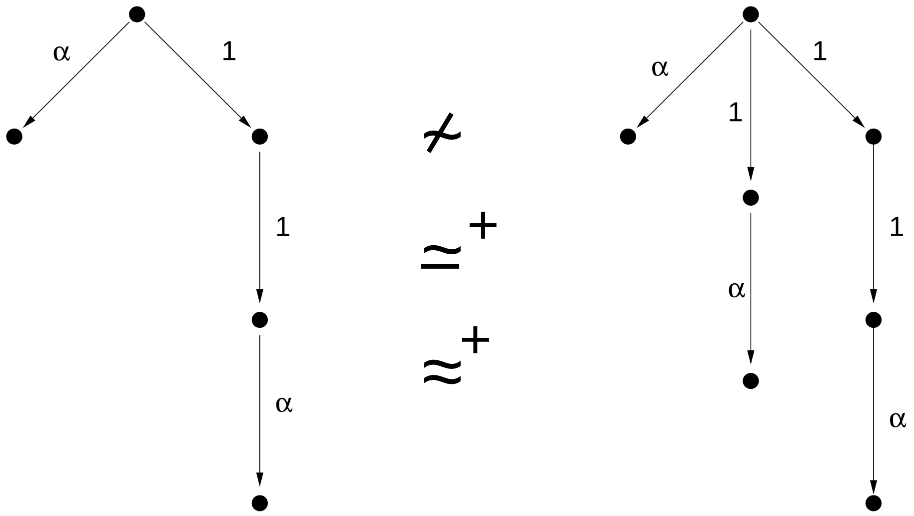

Figure 2: Distinguishing strong and efficiency congruences

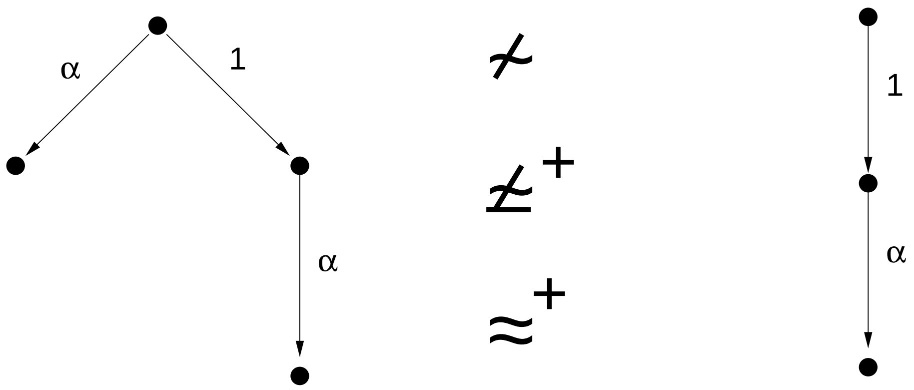

Figure 3: Distinguishing efficiency and observational congruences

4 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 and all occurrences of summation are finitary.

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 $\mathbf{0}$, prefix and the binary summation operations. This is because, any finite process containing one or more of the so called "static operators", viz. composition, hiding, relabelling and disabling may be transformed into a finite serial process using equations which are sound for strong congruence. The reader may verify from the following proposition (stated without proof) that this is indeed true. It is easy to show that " $\mid$ " is both commutative and associative, hence we use the unary operation "Γ" over a set of processes to denote their composition. For similar reasons "$\sum$" is used for summation. Versions of the following proposition have been proved by Milner [10]. The equations relevant to the disabling operator are easy to prove.

Proposition 4.1

1. *The Expansion Law. Let* $p \triangleq \Gamma\tilde{p}$, *where* $\tilde{p} = \{p_i \mid 1 \leq i \leq n\}$ *is a finite set of processes. Then*

$$\begin{aligned} p \quad \sim \quad & \textstyle\sum\{a_i.p' \mid p_i \xrightarrow{a_i} p'_i, 1 \leq i \leq n, p' \equiv \Gamma((\tilde{p} - \{p_i\}) \cup \{p'_i\})\} + \\ & \textstyle\sum\{1.p'' \mid p_i \xrightarrow{\alpha} p'_i, p_j \xrightarrow{\bar{\alpha}} p'_j, 1 \leq i < j \leq n, p'' \equiv \Gamma((\tilde{p} - \{p_i, p_j\}) \cup \{p''_i, p''_j\})\} \end{aligned}$$

2.

$$\begin{array}{lcll} (a.p)\backslash L & \sim & \mathbf{0} & \textit{if } a \in L \cup \bar{L} \\ (a.p)\backslash L & \sim & a.(p\backslash L) & \textit{if } a \notin L \cup \bar{L} \\ (a.p)[h] & \sim & h(a).p[h] & \\ (a.p) \rhd q & \sim & a.(p \rhd q) + q & \end{array}$$

3.

$$\begin{array}{lcl@{\qquad}lcl} \mathbf{0}\backslash L & \sim & \mathbf{0} & (p+q)\backslash L & \sim & (p\backslash L) + (q\backslash L) \\ \mathbf{0}[h] & \sim & \mathbf{0} & (p+q)[h] & \sim & p[h] + q[h] \\ \mathbf{0} \rhd q & \sim & q & (p+q) \rhd r & \sim & (p \rhd r) + (q \rhd r) \\ p \rhd \mathbf{0} & \sim & p & & & \end{array}$$

Note that the last equation viz $p \rhd \mathbf{0} \sim p$ is provable from the other equations. A proof system for efficiency precongruence, strictly speaking, should also incorporate the equations of proposition 4.1 (some of them are infinitary since A is an infinite set). However, it is enough for our purposes to know that such equations exist. Our main concern therefore, will be to obtain a proof system for finite serial processes that characterizes efficiency precongruence. In passing, we also note that the identity $p \rhd \mathbf{0} \sim p$ may be proven by induction on the maximum depth of prefixes in p.

For the purposes of this section we restrict our attention to the following sublanguage of CCS.

$$
\begin{array}{llcl}
\textit{Commutativity} & x + y & = & y + x \\
\textit{Associativity} & x + (y + z) & = & (x + y) + z \\
\textit{Idempotence} & x + x & = & x \\
\textit{Identity} & x + \mathbf{0} & = & x
\end{array}
$$

Figure 4: $\mathcal{A}_{\sim}$: The proof system for strong congruence

$$
\begin{array}{llcl}
\textit{Choice Elimination} & a.(x + 1.x) & \sqsubseteq & a.x \\
\textit{Choice Introduction} & 1.x & \sqsubseteq & x + 1.x \\
\textit{Summand Introduction} & a.(x + 1.y) & \sqsubseteq & a.(x + 1.y) + a.y
\end{array}
$$

Figure 5: $\mathcal{A}_{\tau}$: The new "τ-laws"

$$e ::= \mathbf{0} \mid a.e \mid e + e$$

Our proof system, denoted $\mathcal{A}_{\lesssim^+}$, combines the axioms of $\sim$ (shown in figure 4), the new "τ-laws" $\mathcal{A}_{\tau}$ (shown in figure 5) and an extra rule of inference shown separately as figure 6.

As with any inequational axiom system, provability is assumed to be closed under the rules in figure 7. Further, each of the equations in figure 4, actually denotes a pair of inequations which may be obtained by applying the Equality rules shown in figure 7.

Most of the rules (and axioms) require no explanation. Some of them (the so-called "τ-laws" in figure 5) 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 a slightly weaker version of *Choice Elimination* viz. $a.1.x. \sqsubseteq a.x$ may be derived using *Choice Introduction*. However, rule *Summand Elimination* may require a detailed intuitive explanation to justify its introduction. Let $\mathcal{A}^{-}$ denote the proof system $calA_{\lesssim^+}$ without *Summand Elimination*.

To explain the need for this rule, it is necessary to be able to compare the proof system $\mathcal{A}_{\lesssim^+}$ with the system $\mathcal{A}_{\approx^+}$ for observational congruence. $\mathcal{A}_{\approx^+}$ consists of the four equations of $\mathcal{A}_{\sim}$ and the "*tau*-laws", $\mathcal{A}_{\tau\approx^+}$ shown in figure 8.

In place of *Abstraction* we could have used a form of *Choice Elimination* viz. $a.(x + 1.x) = a.x$, but our choice seems more straightforward. Moreover these are precisely the axioms used in [7]. Further we also have the rules of equational logic which are obtained by replacing all occurrences

$$
\textit{SummandElimination} \quad \frac{x \sqsubseteq x + y + z}{x \sqsubseteq x + z}
$$

Figure 6: The new rule

Reflexivity.	$\dfrac{}{x \sqsubseteq x}$
Transitivity.	$\dfrac{x \sqsubseteq y,\ y \sqsubseteq z}{x \sqsubseteq z}$
Equality.	$\dfrac{x = y}{x \sqsubseteq y},\ \dfrac{x = y}{y \sqsubseteq x},\ \dfrac{x \sqsubseteq y,\ y \sqsubseteq x}{x = y}$
Substitutivity.	$\dfrac{x_1 \sqsubseteq y_1,\ \cdots,\ x_k \sqsubseteq y_k}{\odot(x_1, \dots, x_k) \sqsubseteq \odot(y_1, \dots, y_k)}$ for every k-ary operator $\odot$
Instantiation.	$\dfrac{x \sqsubseteq y}{x\sigma \sqsubseteq y\sigma}$ for every substitution σ

Figure 7: The rules of Inequational logic

Abstraction	$a.1.x$	$=$	$a.x$
Choice Introduction	$1.x$	$=$	$x + 1.x$
Summand Introduction	$a.(x + 1.y)$	$=$	$a.(x + 1.y) + a.y$

Figure 8: The "τ-laws", $\mathcal{A}_{\tau\approx^+}$

of $\sqsubseteq$ by $=$ in figure 7. $\mathcal{A}_{\approx^+}$ has been shown to be complete for finite processes (see [7] or [10]). Now consider the following equation.

$$z + 1.(x + y) = z + 1.(x + y) + y \tag{12}$$

It may be easily shown that (12) is provable in $\mathcal{A}_{\approx^+}$ using the equation

$$1.(x + y) = 1.(x + y) + y. \tag{13}$$

(13) in turn may be derived from *Choice Intruduction* in $\mathcal{A}_{\approx^+}$ (figure (8)) in the following manner.

$$\begin{array}{lll} & 1.(x + y) & \\ = & 1.(x + y) + (x + y) & \textit{Choice Introduction} \\ = & (1.(x + y) + (x + y)) + y & \textit{Idempotence, Associativity} \\ = & 1.(x + y + y & \textit{Choice Introduction} \end{array}$$

However, to prove the analogue of (12), viz.

$$z + 1.(x + y) = z + 1.(x + y) + y \tag{14}$$

in $\mathcal{A}_{\lesssim^+}$, it would be necessary to introduce

$$1.(x+y) \sqsubseteq 1.(x+y)+y. \tag{15}$$

as an extra axiom, since (15) cannot be derived from *Choice Introduction* in figure (5) in a similar fashion. Even if (15) were added to the system $\mathcal{A}_{\lesssim^+}$, it is not possible to prove

$$1.(x_1+1.(x_2+y)) \sqsubseteq 1.(x_1+1.(x_2+y))+y \tag{16}$$

whereas, the analogue of (16), viz.

$$1.(x_1+1.(x_2+y)) = 1.(x_1+1.(x_2+y))+y \tag{17}$$

is provable in $\mathcal{A}_{\approx^+}$, without the addition of any more axioms or inference rules.

Let $\tilde{x}$ denote the sequence $\langle x_i \mid 1 \leq i \leq m \rangle$ and for any y, let $a \nrightarrow \tilde{x} \square y$ denote $a.(x_1 + a.(x_2 + a.(\ldots + a.(x_m+y)\ldots)))$. In general, in the system $\mathcal{A}^-$, it does not seem possible to derive valid inequalities of the form,

$$1 \nrightarrow \tilde{x} \square y \sqsubseteq (1 \nrightarrow \tilde{x} \square y) + y \tag{18}$$

without adding an infinite number of axioms starting with (15). On the other hand, by using the rule *Summand Elimination* in figure (6) in conjunction with *Choice Introduction*, we would be able to prove inequations like (18). For instance, (16) may be derived in $\mathcal{A}_{\lesssim^+}$ as follows.

$$\begin{array}{lll}
 & 1.(x_1+1.(x_2+y)) & \\
\sqsubseteq & 1.(x_1+1.(x_2+y))+x_1+1.(x_2+y) & \textit{Choice Introduction} \\
\sqsubseteq & 1.(x_1+1.(x_2+y))+x_1+(1.(x_2+y)+x_2+y) & \textit{Choice Introduction} \\
\sqsubseteq & 1.(x_1+1.(x_2+y))+(x_1+(1.(x_2+y)+x_2)+y & \textit{Associativity} \\
\sqsubseteq & 1.(x_1+1.(x_2+y))+y & \textit{Summand Elimination}
\end{array}$$

Choice Introduction therefore enables us to extract y from an expression of the form given in (18), in which it may be nested at an arbitrary depth. However *Summand Elimination* allows us to discard certain other superfluous terms which also come up by the application of *Choice Introduction*. A similar phenomenon occurs with *Summand Introduction*. For example,

$$a.(x_1+1.(x_2+1.y)) \sqsubseteq a.(x_1+1.(x_2+1.y))+a.y \tag{19}$$

cannot be derived in $\mathcal{A}^-$ although the corresponding identity is a theorem in $\mathcal{A}_{\approx^+}$.

We give below the proof of soundness of the new rule and leave the other proofs of soundness to the reader. We then proceed to show that the proof system $\mathcal{A}_{\lesssim^+}$ is complete for finite processes.

Proposition 4.2 *(Soundness of Summand Elimination). If $p \gtrsim^+ p+q+r$ then $p \gtrsim^+ p+r$.*

Proof. Let $p \lesssim^+ p+q+r \equiv pqr$ and let $pr \equiv p+r$. By proposition 3.10 we have

$$p \xrightarrow{a} p' \implies \exists pqr' \equiv p' : pqr \xrightarrow{a} pqr' \wedge p' \lesssim pqr'$$

Letting $pr' \equiv p'$ we get

$$p \xrightarrow{a} p' \implies \exists pr' : pr \xrightarrow{a} pr' \wedge p' \lesssim pr'$$

Further we also have from proposition 3.10,

$$pqr \xrightarrow{a} pqr' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p' \lesssim pqr'$$

from which it follows that

$$pr \xrightarrow{a} pr' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p' \lesssim pr'$$

■

Lemma 4.3 *For all processes p, q, $p \lesssim q$ implies $p \lesssim^+ q$ or $p \lesssim^+ q + 1.q$.*

Proof. Assume $p \lesssim q$ and $p \not\lesssim^+ q$. We have to show $p \lesssim^+ q + 1.q$. By proposition 3.10 it is sufficient to show

$$p \xrightarrow{a} p' \implies \exists q' : q + 1.q \xrightarrow{a} q' \wedge p' \lesssim q' \tag{20}$$

$$q + 1.q \xrightarrow{a} q' \implies \exists p' : p \overset{a}{\Longrightarrow} p' \wedge p' \lesssim q' \tag{21}$$

(20) follows from the fact that $p \lesssim q$ using the characterization of $\lesssim$ given in proposition 3.5. The same characterization ensures (21) whenever $q \xrightarrow{a} q'$. Hence we only need to show $\exists p' : p \xrightarrow{1} p' \wedge p' \lesssim q'$ when $q + 1.q \xrightarrow{1} q$. Since $p \lesssim q$ and $p \not\lesssim^+ q$, the condition (7) and the negation of condition (9) for $a = 1$ yield

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

Hence $q + 1.q \xrightarrow{1} q$ implies $\exists p' : p \overset{1}{\Longrightarrow} p' \wedge p' \lesssim q$ which was required. ■

Lemma 4.4 $p \xrightarrow{a} p'$ *implies* $\mathcal{A}_{\lesssim^+} \vdash p = p + a.p'$.

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

Lemma 4.5 $p \overset{a}{\Longrightarrow} p'$ *implies* $\mathcal{A}_{\lesssim^+} \vdash p \sqsubseteq p + a.p'$.

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

$$\begin{array}{llllll}
\mathcal{A}_{\lesssim^+} & \vdash & p & = & p+1.q & \\
\mathcal{A}_{\lesssim^+} & \vdash & q & \sqsubseteq & q+a.p' & \textit{Induction hypothesis} \\
\mathcal{A}_{\lesssim^+} & \vdash & 1.q & \sqsubseteq & 1.(q+a.p') & \textit{Substitutivity} \\
\mathcal{A}_{\lesssim^+} & \vdash & 1.q & \sqsubseteq & 1.(q+a.p')+q+a.p' & \textit{Choice Introduction} \\
\mathcal{A}_{\lesssim^+} & \vdash & p & \sqsubseteq & (1.(q+a.p')+q)+a.p' & \textit{Transitivity} \\
\mathcal{A}_{\lesssim^+} & \vdash & p & \sqsubseteq & p+a.p' & \textit{Summand Elimination}
\end{array}$$

In the latter case, by the induction hypothesis

$$\begin{array}{llllll}
\mathcal{A}_{\lesssim^+} & \vdash & p & \sqsubseteq & p+a.q & \textit{Induction Hypothesis} \\
\mathcal{A}_{\lesssim^+} & \vdash & q & = & q+1.p' & \textit{the previous lemma} \\
\mathcal{A}_{\lesssim^+} & \vdash & a.q & = & a.(q+1.p') & \textit{Substitutivity} \\
\mathcal{A}_{\lesssim^+} & \vdash & a.q & \sqsubseteq & a.(q+1.p')+a.q & \textit{Summand Introduction} \\
\mathcal{A}_{\lesssim^+} & \vdash & p & \sqsubseteq & p+a.(q+1.p')+a.q & \textit{Transitivity} \\
\mathcal{A}_{\lesssim^+} & \vdash & p & \sqsubseteq & p+a.p' & \textit{Summand Elimination}
\end{array}$$

■

Theorem 4.6 *(Completeness). $p \lesssim^+ q$ implies $\mathcal{A}_{\lesssim^+} \vdash p \sqsubseteq q$.*

Proof. We prove this by induction on the *sum*[2] 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

$$p \equiv \sum\{a_i.p_i \mid 1 \leq i \leq m\}$$

and

$$q \equiv \sum\{b_j.q_j \mid 1 \leq j \leq n\}$$

respectively, and for all i, j, $1 \leq i \leq m$, $1 \leq j \leq n$, 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 *Identity*, all summands that are $\mathbf{0}$ may be eliminated leaving p and q in standard form $(m, n > 0)$. We now prove the following claims.

Claim 1 $\mathcal{A}_{\lesssim^+} \vdash a_i.p_i + q \sqsubseteq q$, for all i, $1 \leq i \leq m$.

[2] In most proofs of completeness of equational axiomatizations for finite serial processes, the *maximum* is used. However, as we shall see that will not work in this case.

If $p \lesssim^{+} q$ then since $p \xrightarrow{a_i} p_i$, there exists $b_j = a_i$ with $q \xrightarrow{b_j} q_j$ and $p_i \lesssim q_j$. By lemma 4.3 $p_i \lesssim^{+} q_j$ or $p_i \lesssim^{+} q_j + 1.q$. By the induction hypothesis we have $\mathcal{A}_{\lesssim^{+}} \vdash p_i \sqsubseteq q_j$ or $\mathcal{A}_{\lesssim^{+}} \vdash p_i \sqsubseteq q_j + 1.q_j$. In the former case $\mathcal{A}_{\lesssim^{+}} \vdash a_i.p_i \sqsubseteq b_j.q_j$ by *Substitutivity*, and in the latter case we get the same result by applying *Substitutivity* followed by *Choice Elimination*. The result then follows.

<u>*Claim 2*</u> $\mathcal{A}_{\lesssim^{+}} \vdash p \sqsubseteq p + b_j.q_j$, for all j, $1 \leq j \leq n$.

If $q \xrightarrow{b_j} q$ then by proposition 3.10 there exists p' and some $a_i = b_j$ such that $p \xRightarrow{a_i} p'$ and $p' \lesssim q_j$. By lemma 4.3 and the induction hypothesis we get $\mathcal{A}_{\lesssim^{+}} \vdash p' \sqsubseteq q_j$ or $\mathcal{A}_{\lesssim^{+}} \vdash p' \sqsubseteq q_j + 1.q_j$. It then follows (by applying *Choice Elimination* in the latter case) that $\mathcal{A}_{\lesssim^{+}} \vdash b_j.p' \sqsubseteq b_j.q_j$ and by lemma 4.5 $\mathcal{A}_{\lesssim^{+}} \vdash p \sqsubseteq p + b_j.p'$ from which we obtain $\mathcal{A}_{\lesssim^{+}} \vdash p \sqsubseteq p + b_j.q_j$.

Having proved the claims, we proceed as follows. By *Substitutivity*, we may sum up (4) for all i, reorder the terms and repeatedly apply *Idempotence* to obtain $\mathcal{A}_{\lesssim^{+}} \vdash p + q \sqsubseteq q$. By performing similar operations with (4) for all j, we get $\mathcal{A}_{\lesssim^{+}} \vdash p \sqsubseteq p+q$.The result then follows by *Transitivity*. ■

5 A Long Example: The (N+2)-Buffer

In this section we work out in detail various examples of specifications and implementations. It should be understood that for reasons of clarity and convenience we deviate from the notational conventions specified in section 2. Further we use definitions by case analysis on the state of a process. The standard mechanism in value-passing CCS is to use a conditional expression. However, we prefer to write them in a more informal but mathematically acceptable style. Some notational conventions that we follow in this section are

- the symbol '$\triangleq$' is used to denote a definition of a process,
- process names are usually written in uppercase and
- action names are usually written in `typewriter` font.
- Γ will be used to denote the composition of a finite set of processes (notice that $|$ is commutative and associative).

In the example we use the value-passing version to specify a bounded buffer of capacity $N+2$ (the reason for choosing this number will become clear from the example) 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 of acceptance 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 under $\lesssim$.

The specification FIFO. Let D be a value domain and for $n \geq 0$, let D^n denote the set of strings of length n and let $D_n = \bigcup\{D^k \mid 0 \leq k \leq n\}$. The state of the buffer at any instant of

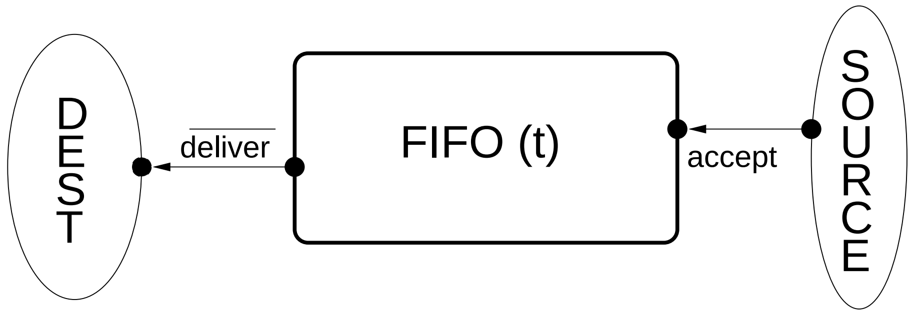

Figure 9: The (N+2)-Buffer specification

its execution is determined by the string it contains. The state space of the buffer is given by D_{N+2}. For any $t \in D_{N+2}$, $|t|$ denotes the length of the string t. We may then formally specify the behaviour of the buffer by the CCS process $FIFO(t)$ (see figure 9), where $t \in D_{N+2}$ as follows.

$$FIFO(t) \triangleq \begin{cases} \mathtt{accept}(a).FIFO(a) & \text{if } |t| = 0 \\ \overline{\mathtt{deliver}}(d).FIFO(t') + \mathtt{accept}(a).FIFO(ta) & \text{if } t = dt' \ \&\ 0 < |t| < N+2 \\ \overline{\mathtt{deliver}}(d).FIFO(t') & \text{if } t = dt' \ \&\ |t| = N+2 \end{cases}$$

For any string $t \in D_{N+2}$, $FIFO(t)$ has the following transitions.

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

$$FIFO(t) \xrightarrow{\overline{\mathtt{deliver}}(d)} FIFO(t') \quad \text{if } 0 < |t| \leq N+2 \text{ and } t = dt'$$

Throughout this section we shall use the word "implementation" to refer to any CCS process that is observationally congruent to $FIFO(\varepsilon)$.

The implementation PIPE. A simple implementation of this buffer is in terms of cells, where a cell is an i-o device that may be defined as $C \triangleq \mathtt{in}(x).C'(x)$ and $C'(x) \triangleq \overline{\mathtt{out}}(x).C$. For the sake of uniformity we denote the state C as $C(\perp)$ and $C'(d)$ as $C(d)$. A buffer of capacity $N+2$ may be created simply by connecting cells end to end and relabelling the ports appropriately (see figure 10).

Therefore we have for $D_\perp = D \cup \{\perp\}$ and $x \in D_\perp$,

$$\begin{array}{lcl} C_0(x_0) & \triangleq & C(x_0)[\mathtt{fwd}_0/\mathtt{in}, \mathtt{deliver}/\mathtt{out}] \\ C_j(x_j) & \triangleq & C(x_j)[\mathtt{fwd}_j/\mathtt{in}, \mathtt{fwd}_{j-1}/\mathtt{out}], \text{ for } 0 < j < N+1 \\ C_{N+1}(x_{N+1}) & \triangleq & C(x_{N+1})[\mathtt{accept}/\mathtt{in}, \mathtt{fwd}_N/\mathtt{out}] \\ PIPE(s) & \triangleq & (\Gamma\{C_j(x_j) \mid 0 \leq j < N+2\}) \backslash \{\mathtt{fwd}_k \mid 0 \leq k < N+1\} \end{array}$$

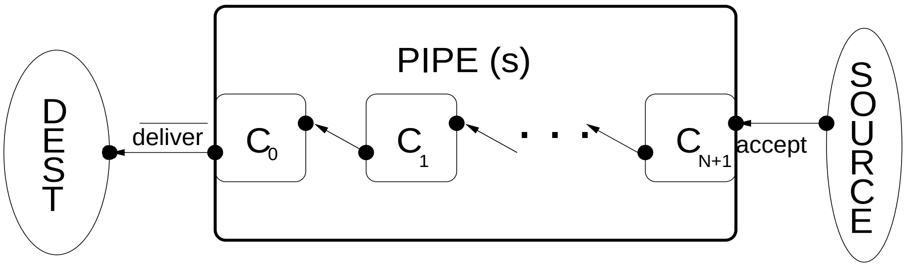

Figure 10: The implementation PIPE

where $s = x_0 \cdots x_{N+1} \in (D_\perp)^{N+2}$. The transitions of $PIPE(s)$ are

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

$$PIPE(ds') \xrightarrow{\overline{\texttt{deliver}}(d)} PIPE(\perp s') \quad s' \in (D_\perp)^{N+1}, d \in D$$

$$PIPE(s'\perp vs") \xrightarrow{1} PIPE(s'v\perp s"), \qquad s's" \in (D_\perp)^N, v \in D$$

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 5.1 *Let $f : (D_\perp)_{N+2} \rightarrow D_{N+2}$ be the function which strips off all occurrences of $\perp$ from any string in $(D_\perp)_{N+2}$. Then*

1. $R = \{\langle PIPE(s), FIFO(f(s))\rangle \mid s \in (D_\perp)^{N+2}\}$ *is an ep.*
2. $PIPE(\perp^{N+2}) \precsim^{+} FIFO(\varepsilon)$.

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 $PIPE(\perp^{N+2})$ has no silent transitions. Then $PIPE(s)$ and $FIFO(t)$ satisfy the conditions 8 and 9. Hence $PIPE(\perp^{N+2}) \precsim^{+} FIFO(\varepsilon)$.

■

The implementation BUFF. Now consider a different implementation of the buffer, in which N cells are used (numbered from 0 to $N-1$) simply as storage and each cell interacts only with

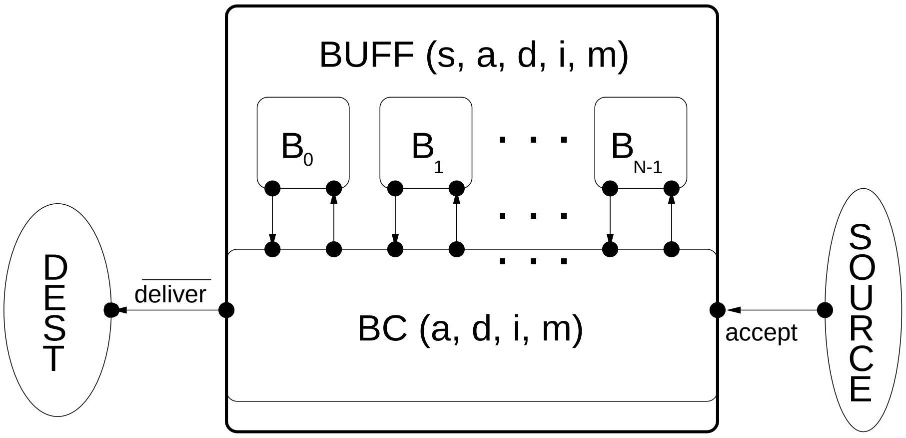

Figure 11: The implementation BUFF

a centralized buffer controller which may store an additional two values. We define the store as follows.

$$\begin{array}{lcl} B_j(x) & \triangleq & C(x)[\mathtt{write}_j/\mathtt{in}, \mathtt{read}_j/\mathtt{out}], \quad \text{for } 0 \leq j < N \\ MEM(s) & \triangleq & \Gamma\{B_j(x_j) \mid 0 \leq j < N\} \end{array}$$

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

The buffer controller, BC, whose schematic design is shown in figure 11 may hold at most two values at any instant. It performs the following functions.

- it *accepts* a value from $SOURCE$ through the port `accept`, and does not accept another till the first one is stored in one of the N cells.
- it *writes* the most recently accepted value in the first available empty cell, say B_j, through the port $\overline{\mathtt{write}}_j$.
- it retains the oldest undelivered value, (after *reading* from, say B_i through port $\mathtt{read}_i$) and keeps it ready for delivery.
- it *delivers* whatever value it has read in, whenever possible, to $DEST$, through the port $\overline{\mathtt{deliver}}$. The next value is not delivered till it has been retrieved from the appropriate cell.
- the N cells are treated as a circular queue of length N (with the indices of the cells ordered as follows: $0 < 1 < \cdots < N-1 < 0$). It is clear that the controller requires to maintain the following information:
  - i, the index of the cell containing the oldest undelivered message,
  - 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.

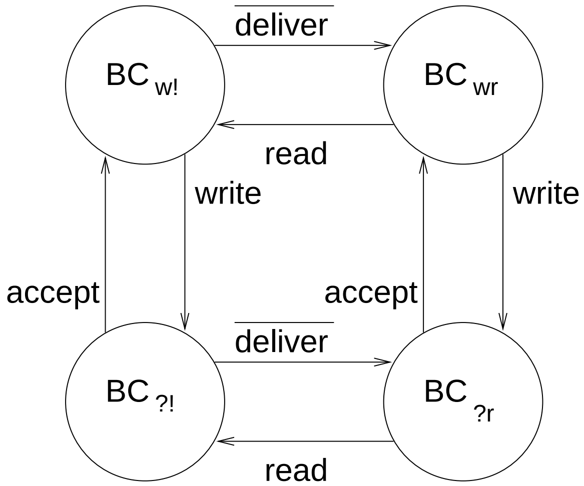

Figure 12: Transition diagram of BC when MEM is neither full nor empty

The state of BC is therefore determined by four arguments – the value $a \in D$ at the port `accept`, which it may accept from $SOURCE$, the value $d \in D$ at the port $\overline{\texttt{deliver}}$, which it may deliver to $DEST$, the index i and the number m. As before $a = \perp$ if there is no value at `accept` and similarly $d = \perp$ if there is no value at `deliver`. Further, if n denotes the total number of messages in the memory-buffer controller system, then $m \leq n \leq m+2$ 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 $DEST$ $(n = m + 1)$

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

Figure 12 shows the transitions that BC may go through between these four states provided MEM continues to be neither empty nor full. When $m = 0$ and $n = 2$, clearly the only kind of state possible is $BC_{w!}$. Similarly, when MEM is full (i.e. $N = m < n \leq N + 2$), 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" is) the possible

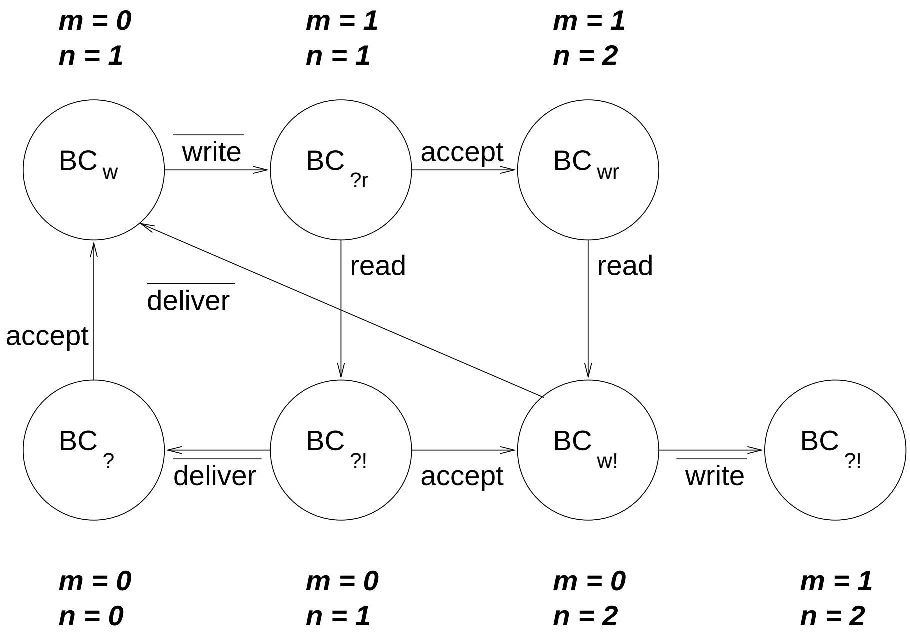

Figure 13: Transition diagram of BC when MEM is empty

kinds of states that BC could be in are $BC_{?!}$, $BC_{w!}$, BC_{wr}. The state transition diagrams for these "boundary" conditions are shown in figures 13 and 14 respectively. We may then define $BC(a, d, i, m)$ as follows, using "$\oplus$" and "$\ominus$" to denote addition and subtraction modulo N respectively.

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

where

$$BC_{?}(\perp,\perp,i,0) \triangleq \mathtt{accept}(a).BC_{w}(a,\perp,i,0)$$

$$BC_{w}(a,\perp,i,0) \triangleq \overline{\mathtt{write}}_i(a).BC_{?r}(\perp,\perp,i,1)$$

$$BC_{?!}(\perp,d,i,m) \triangleq \begin{cases} \mathtt{accept}(a).BC_{w!}(a,d,i,m) + \\ \quad \overline{\mathtt{deliver}}(d).BC_{?}(\perp,\perp,i,m) & \text{if } m = 0 \\ \mathtt{accept}(a).BC_{w!}(a,d,i,m) + \\ \quad \overline{\mathtt{deliver}}(d).BC_{?r}(\perp,\perp,i,m) & \text{if } 0 < m < N \\ \mathtt{accept}(a).BC_{!}(a,d,i,m) + \\ \quad \overline{\mathtt{deliver}}(d).BC_{?r}(\perp,\perp,i,m) & \text{if } m = N \end{cases}$$

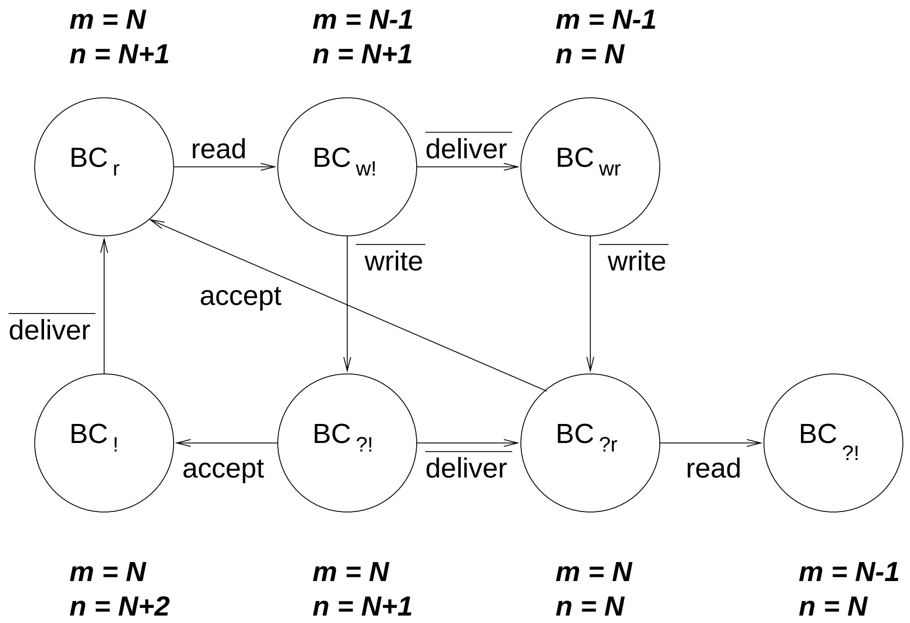

Figure 14: Transition diagram of BC when MEM is full

$$BC_{w!}(a,d,i,m) \triangleq \begin{cases} \overline{\texttt{write}}_{i\oplus m}(a).BC_{?!}(\bot,d,i,m+1) + \\ \quad \overline{\texttt{deliver}}(d).BC_w(a,\bot,i,m) & \text{if } m = 0 \\ \overline{\texttt{write}}_{i\oplus m}(a).BC_{?!}(\bot,d,i,m+1) + \\ \quad \overline{\texttt{deliver}}(d).BC_{wr}(a,\bot,i,m) & \text{if } 0 < m < N \end{cases}$$

$$BC_{?r}(\bot,\bot,i,m) \triangleq \begin{cases} \texttt{accept}(a).BC_{wr}(a,\bot,i,m) + \\ \quad \texttt{read}_i(d).BC_{?!}(\bot,d,i\oplus 1,m-1) & \text{if } 0 < m < N \\ \texttt{accept}(a).BC_r(a,\bot,i,m) + \\ \quad \texttt{read}_i(d).BC_{?!}(\bot,d,i\oplus 1,m-1) & \text{if } m = N \end{cases}$$

$$BC_{wr}(a,\bot,i,m) \triangleq \overline{\texttt{write}}_{i\oplus m}(a).BC_{?r}(\bot,\bot,i,m+1) \ + \texttt{read}_i(d).BC_{w!}(a,d,i\oplus 1,m-1)$$

$$BC_r(a,\bot,i,N) \triangleq \texttt{read}_i(d).BC_{w!}(a,d,i\oplus 1,N-1)$$

$$BC_!(a,d,i,N) \triangleq \overline{\texttt{deliver}}(d).BC_r(a,\bot,i,N)$$

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 its 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. Now the system $BUFF$ (with an appropriate subscript reflecting the state of BC) may be defined as follows.

$$BUFF_*(s,a,d,i,m) \triangleq (MEM(s) \mid BC_*(a,d,i,m))\backslash\{\texttt{read}_k, \texttt{write}_k \mid 0 \le k < N\}$$

where $* \in \{?, !, w, r, ?!, w!, ?r, wr\}$.

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.

$$BUFF(s, \perp, d, i, m) \xrightarrow{\texttt{accept}(a)} BUFF(s, a, d, i, m) \qquad \text{if } a \in D,\ d \in D_\perp$$

$$BUFF(s, a, d, i, m) \xrightarrow{\overline{\texttt{deliver}}(d)} BUFF(s, a, \perp, i, m) \qquad \text{if } a \in D_\perp,\ d \in D$$

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

The last transition given above holds because the ports $\texttt{read}_k$, $\overline{\texttt{read}_k}$, $\texttt{write}_k$ and $\overline{\texttt{write}_k}$, for all k, $0 \leq k < N$, are hidden. The restriction 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.

To be able to prove that this system works correctly we require the following function

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

that 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.

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

where x_j is the value stored in cell B_j , for all $i \leq j < i \oplus m$. We now have the following lemma.

Lemma 5.2

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, $0 \leq k < m$, $x_{i\oplus k} = y_{j\oplus k}$.*
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 $BUFF_*(s, a, d, i, m) \simeq^+ BUFF(s', a, d, i', m)$. 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 restriction operation $\backslash\{\texttt{read}_k, \texttt{write}_k \mid 0 \leq k < N\}$ 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 above, that the relation R,

   $$R = \{\langle BUFF_*(s, a, d, i, m), BUFF_*(s', a, d, i', m)\rangle \mid g(s, i, m) = g(s', i', m)\}$$

   is a strong bisimulation.

■

As a consequence of the above lemma we may abbreviate the state of $BUFF$ to

$$BUFF_*(dra) \equiv BUFF_*(s, a, d, i, m)$$

where $r = g(s, i, m) \in D_N$ and the subscript $*$ determines the state of the buffer controller BC. This means for $r \in D_N$ and $a, d \in D_\perp$, we have

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

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

$$BUFF_*(t) \equiv BUFF_*(dra)$$

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

Proposition 5.3 *Let* $S = \{\langle BUFF_*(t), FIFO(t)\rangle \mid t \in D_{N+2}\}$.

1. *S is an efficiency prebisimulation.*
2. $BUFF_*(\varepsilon) \precsim^+ FIFO(\varepsilon)$.

Proof outline.

1. Follows from the transitions of $FIFO$ and $BUFF$.
2. When $t = \varepsilon$, $* = ?$ and $BUFF_?(t)$ has no internal transitions. In fact, both $BUFF(\varepsilon)$ and $FIFO(\varepsilon)$ have only an `accept`-transition, satisfying the conditions 8 and 9 of proposition 3.10.

■

Proposition 5.4 *Let* $T = \{\langle PIPE(dsa), BUFF(dra)\rangle \mid f(s) = r, a, d \in D_\perp\}$.

1. *T is an efficiency prebisimulation for all* $N > 0$.
2. *For* $N = 1$ *and* $s, r \in D_\perp$ *such that* $f(s) = r$, $PIPE(dsa) \sim BUFF(dra)$ *for all* $a, d \in D_\perp$.
3. *For* $N > 1$ *and* $s, r \in (D_\perp)_N$ *such that* $f(s) = r$, $PIPE(dsa) \precsim^+ BUFF(dra)$ *for all* $a, d \in D_\perp$. *In particular,* $PIPE(\perp^{N+2}) \precsim^+ BUFF_?(\varepsilon)$.

Proof outline.

1. Similar to the proof of the first part of proposition 5.3. The condition $N > 0$ 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 is actually a strong bisimulation.
3. It suffices to consider only the internal transitions of the 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 *write* the message in MEM and the other to *read* it from MEM) before delivering it to $DEST$. Consider any ordered pair $\langle PIPE(dsa), BUFF(dra)\rangle$ in the ep T and let $N > 1$. The following claims may then be proved easily.

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

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

   Finally, $PIPE(\perp^{N+2}) \precsim^{+} BUFF_{?}(\varepsilon)$ follows from the fact that both processes have no internal transitions and both may only perform `accept`-transitions.

■.

The implementation $BUFF$ even though better than $PIPE$, suffers from two drawbacks.

- Firstly, for $N = 0$, it is not even an implementation of $FIFO$. It would be nicer to have a buffer of capacity 2 by simply eliminating the memory.
- Secondly, the implementation could be optimized for the case when $m = 0$. Specifically when the memory is empty (for example in the initial state) it should be possible for a message that is *accepted* to be made ready for delivery without actually first *writing* it into memory.

The correction of the second drawback is quite simple, we need to change our definition of $BC_{?}(\perp, \perp, i, 0)$ to

$$BC_{?}(\perp, \perp, i, 0) \triangleq \texttt{accept}(a).BC_{?!}(\perp, a, i, 0)$$

It turns out that by removing all states and clauses in the definition of of BC that relate to *reading* and *writing* of the memory, this impelmentation then reduces (upto strong congruence) to the specification for $N = 0$.

6 Conclusion

The efficiency preorder we have introduced, 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, [10], 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, [4], may easily be adapted to $\precsim$. 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: $p \precsim q$ 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 implementations 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 $\precsim$ completely using a finite set of inequations. Recall that the proof rule *Summand Elimination* is essential to our proof system.

There has been much recent work on introducing notions of time into process algebras. [3] and [12] are typical examples of one approach, where real-time durations are associated with actions. [11] 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.

There are other reasons for considering this work to be important. One of them is that the efficiencies of systems with infinite behaviour has never been addressed by the algorithms community. The community so far has used a lot of powerful mathematics to address the complexity of "parts of systems" rather than whole systems.

Another reason why this work may be considered important is that so far efficiency and correctness have always been considered separately. Any analysis of an algorithm first proves the correctness of the algorithm in the first phase and then its complexity is analysed in a separate phase. We have however shown that it is at least feasible in the case of concurrent systems to analyse both relative efficiency and correctness within the same semantic framework.

References

[1] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. In *Theoretical Aspects of Computer Software, Sendai 1991*, number 526 in Lecture Notes in Computer Science, pages 152–175. Springer-Verlag, 1991.

[2] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[3] J. Baeton and J. Bergstra. Real time process algebra. Technical Report, CWI Amsterdam, 1989.

[4] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based verification tool for finite-state systems. In *Proceedings of the Workshop on Automated Verification Methods for Finite-state Systems*, volume 29, 1989.

[5] R. Gerth and A. Boucher. A timed failures model for extended communicating sequential processes. In *??*, volume 267. Lecture Notes in Computer Science, 1992.

[6] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[7] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[8] M. Hennessy and T. Regan. A temporal process algebra. Technical Report, 2/90, University of Sussex, 1990.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[10] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[11] X. Nicollin, J.L. Richier, J. Sifakis, and J. Voiron. Atp: An algebra for timed processes. Technical Report, Grenoble, 1989.

[12] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. In *??*, volume 226. Lecture Notes in Computer Science, 1986.

[13] S. Rudkin and C.R. Smith. A temporal enhancement for LOTOS. Handwritten notes, 1990.

[14] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In *CONCUR '92.* Lecture Notes in Computer Science, 1992.