INFORMATION AND COMPUTATION 116, 26–37 (1995)

CCS with Priority Choice[1]

JUANITO CAMILLERI

Department of Computer Studies, University of Malta, University Heights, Msida, Malta G.C.

AND

GLYNN WINSKEL

Computer Science Department, Aarhus University, Ny Munkegade, 8000 Aarhus C, Denmark

This paper investigates an extension of Milner's CCS with a priority choice operator called prisum: this operator is very similar to the PRIALT construct of Occam. The new binary prisum operator only allows execution of its second component in the case where the environment is not ready to allow the first component to proceed. This dependency on the set of actions the environment is ready to perform goes beyond that encountered in traditional CCS. Its expression leads to a novel operational semantics in which transitions carry ready-sets (*of the environment*) as well as the normal action symbols from CCS. A notion of strong bisimulation is defined on agents with priority via this semantics. It is a congruence and satisfies new equational laws (including a new expansion law) which are shown to be complete for finite agents with prisum. The laws are conservative over agents of traditional CCS.

1. INTRODUCTION

This paper augments Milner's CCS with a treatment of priority of the kind found in programming languages such as Occam [13]. In Occam, for example, a typical PRIALT construction takes the form

$$\begin{aligned}&\text{PRIALT}\\ &\quad a_1\,?\,X_1 \to P_1\\ &\quad a_2\,?\,X_2 \to P_2\\ &\quad\ \vdots\\ &\quad a_k\,?\,X_k \to P_k.\end{aligned}$$

This represents an agent which awaits the receipt of values at any of the channels $a_1, ..., a_k$, but in such a way as to give priority to communications at channels earlier in the list; a value will only be received at channel a_i if the environment of the process is ready to send at a_i and not ready to send at the channels $a_1, ..., a_{i-1}$. We remark that an alternative in a PRIALT construction need not always have the form $a\,?\,X \to P$. It might instead be $SKIP \to P$; this preempts all later alternatives and so is generally used as the final alternative in a PRIALT construction.

[1] Both authors acknowledge the partial support of Esprit Basic Research Action CEDISYS.

Consider the system shown in Fig. 1 [9]. A controller C is required to repeatedly choose, nondeterministically, to perform either a *tick* action or a *tock* action, provided a *shut-down* action has not been performed by a sensor S. As soon as the sensor performs a *shut-down* action, C should stop performing both *tick* and *tock* actions. In real applications, the agent C could be a control unit regulating a chemical reaction in a chemical plant, a computer system managing a number of resources, etc., and S might represent a sensor that detects excess temperature, a printer fault, etc. The following is an attempt to define the behaviour of the system described above using the operators of CCS:

$$\begin{aligned}SYS &\stackrel{\text{def}}{=} (C \mid S)\backslash i\\ S &\stackrel{\text{def}}{=} \textit{shut-down}.\bar{\imath}.\mathbf{0}\\ C &\stackrel{\text{def}}{=} \textit{tick}.C + \textit{tock}.C + i.\mathbf{0}.\end{aligned}$$

As defined above, SYS may perform a series of *tick* or *tock* actions. As soon as S performs a *shut-down* to become $i.\mathbf{0}$, the sensor is willing to handshake with the controller. Due to the nondeterminism of the $+$ operator, however, the controller is not compelled to handshake with the sensor. In fact, contrary to what is required, *tick tock shut-down tick tick*... is a sequence of actions that may take place in SYS. Even if the semantics of parallel composition is *fair* [10], there is no guarantee that the handshake between sensor and controller will occur immediately.

Suppose the prisum operator $\twoheadrightarrow$ [6] is added to the language of expressions; $\twoheadrightarrow$ gives precedence to the next possible actions of the left operand over the next possible actions of the right operand. For example, the expression $a.E_0 \twoheadrightarrow b.E_1$ can perform a b action only if its environment is not ready to perform an $\bar{a}$ action. Suppose the behaviour of the system illustrated in Fig. 1 is redefined as follows:

$$\begin{aligned}SYS &\stackrel{\text{def}}{=} (C \mid S)\backslash i\\ S &\stackrel{\text{def}}{=} \textit{shut-down}.\bar{\imath}.\mathbf{0}\\ C &\stackrel{\text{def}}{=} i.\mathbf{0} \twoheadrightarrow (\textit{tick}.C + \textit{tock}.C).\end{aligned}$$

0890-5401/95 $6.00

FIG. 1. A simple example.

Before S performs a *shut-down* action, C may perform a *tick* or a *tock* action nondeterministically. As soon as S performs a *shut-down* to become $S' \equiv \bar{\imath}.\mathbf{0}$, S' is ready to perform an $\bar{\imath}$ action. The biased choice operator then forces C to handshake with S' immediately. Therefore when the controller and sensor can handshake, the controller is prohibited from performing any more *tick* or *tock* actions.

This work arose as a continuation of [5], which presented an operational semantics of Occam. We know of no other successful attempts to both semantically define and, with this as a basis, provide a complete proof system for a CCS-like language with a prisum operator. We contribute a new set of laws for agents with priority, complete for finite agents, as part of a well-rounded and, we believe, convincing theory. Claus Torp Jensen has recast the theory presented here with an eye to efficient implementation, and has augmented the Edinburgh–Sussex Concurrency Workbench with tools to check strong bisimulation and to support model checking of CCS agents with priority choice [14].

There have been other attempts to give a semantic basis to reasoning about priority [1–3, 9, 11, 12, 17, 18] (see [6] for more details). These attempts fall under two main categories:

- those that associate priority with choice [1, 5, 7, 17, 18], and
- those that associate priority with events [2, 9, 11].

The prisum operator presented here falls under the first category; it

- allows the environment to resolve a prioritized choice between actions,
- allows global dynamic ordering of actions,
- allows local partial ordering of actions,
- reflects syntactically the priority structure of actions in each state, and
- lends itself to implementation—it is closely related to existing priority constructs in programming languages.

2. CCS WITH PRISUMS

Suppose we were simply to adjoin the prisum operator $\twoheadleftarrow$ to CCS. We would then allow an expression such as $((a.\mathbf{0} \twoheadleftarrow \bar{b}.\mathbf{0}) \mid (b.\mathbf{0} \twoheadleftarrow \bar{a}.\mathbf{0}))\backslash\{a, b\}$. Can this agent perform a τ action? One can argue both ways. Partly to avoid deciding such questions, and partly to help in getting a normal form later, we will avoid a mix of actions and their complements as initial actions of a prisum. We will thus break the symmetry in CCS that exists between actions a and their complements $\bar{a}$, and distinguish between i-actions and o-actions. This distinction is consistent with Occam, where i-actions can be understood as inputs from a channel and o-actions as outputs:

- An i-action is a named action in $\mathscr{A}$ with typical elements $a, b, c, \ldots$,
- an o-action is a conamed action in $\bar{\mathscr{A}}$ with typical elements $\bar{a}, \bar{b}, \bar{c}, \ldots$,
- an internal action is represented by τ.

We shall denote by Act the set of *actions* $\mathscr{A} \cup \bar{\mathscr{A}} \cup \{\tau\}$. We shall understand the operation $\bar{(\ )}$ to act so that $a \mapsto \bar{a}$, $\bar{a} \mapsto a$ and, for technical reasons to be explained later, take $\tau \mapsto *$, where $*$ is a new symbol distinct from any action. (The latter decision is taken so that we can treat τ as being like an i-action.)

CCS augmented with the prisum operator $\twoheadleftarrow$ is called CCS$^{\twoheadleftarrow}$. The syntax of CCS$^{\twoheadleftarrow}$ expressions is defined as follows:

$$\begin{aligned} G &::= X \mid \mathbf{0} \mid a.E \mid \tau.E \mid G\backslash L \mid G[f] \mid \\ &\qquad G + G \mid fix(X = G) \\ AE &::= G \mid AE \twoheadleftarrow AE \\ E &::= X \mid \mathbf{0} \mid \alpha.E \mid E\backslash L \mid E[f] \mid \\ &\qquad E + E \mid E \mid E \mid fix(X = E) \mid AE. \end{aligned}$$

We shall refer to these syntactic sets as $\mathscr{G}^{\twoheadleftarrow}$, $\mathscr{AE}^{\twoheadleftarrow}$, $\mathscr{E}^{\twoheadleftarrow}$ respectively. The set of terms in $\mathscr{E}^{\twoheadleftarrow}$ contains almost all the terms of pure CCS. There are a few restrictions:

- the relabelling function f is injective, and maps $\bar{\mathscr{A}}$ to $\bar{\mathscr{A}}$ and $\mathscr{A}$ to $\mathscr{A}$,
- the set L in $E\backslash L$ is a subset of $\mathscr{A}$, and finally,
- recursion is assumed to be guarded.

Term variables are denoted by X and terms in $\mathscr{E}^{\twoheadleftarrow}$ are ranged over by E, F, while P, Q range over the set of closed terms $\mathscr{P}^{\twoheadleftarrow}$.

3. OPERATIONAL SEMANTICS

The behaviour of CCS$^{\twoheadleftarrow}$ agents is formalised by a transition relation $\vdash_R E \xrightarrow{\alpha} E'$ to be understood as meaning: In an environment which is ready to perform precisely the o-actions R, the agent E can perform an action α to become the agent E'. The transition relation will be defined in terms of two closely related functions, the *acceptance* and *ready* functions. The acceptance function will express those i-actions an agent AE can accept in some environment,

$\mathbf{re}(i)$ $\mathbf{0}$ re $\emptyset$ $\qquad$ $\mathbf{re}(ii)$ $a.E$ re $\emptyset$

$\mathbf{re}(iii)$ $\overline{a}.E$ re $\{\overline{a}\}$ $\qquad$ $\mathbf{re}(iv)$ $\tau.E$ re $\emptyset$

$$\mathbf{re}(v)\ \frac{E_0\ \mathbf{re}\ R_0 \quad E_1\ \mathbf{re}\ R_1}{E_0 + E_1\ \mathbf{re}\ R_0 \cup R_1} \qquad \mathbf{re}(vi)\ \frac{E\ \mathbf{re}\ R}{E \setminus L\ \mathbf{re}\ R - \overline{L}}\ (\text{Recall we assume } L \subseteq \mathcal{A})$$

$$\mathbf{re}(vii)\ \frac{E\ \mathbf{re}\ R}{E[f]\ \mathbf{re}\ f(R)} \qquad \mathbf{re}(viii)\ \frac{E[fix(X = E)/X]\ \mathbf{re}\ R}{fix(X = E)\ \mathbf{re}\ R}$$

$$\mathbf{re}(ix)\ \frac{E\ \mathbf{re}\ R_0 \quad F\ \mathbf{re}\ R_1}{E|F\ \mathbf{re}\ R_0 \cup R_1} \qquad \mathbf{re}(x)\ AE \twoheadrightarrow AF\ \mathbf{re}\ \emptyset$$

FIG. 2. The definition of the ready function **re**.

while the ready function yields the set of o-actions an agent E can do in any environment. They are defined by the rules of Fig. 2 and Fig. 3.

The rules defining the transition relation are presented in Fig. 4. We comment on the rules for prisum and composition. The rule $\mathbf{Pri}_b$ is puzzling, perhaps. A first, though incorrect attempt, might instead be

$$\frac{AE\ \mathbf{ac}\ A \quad \vdash_R \ AF \xrightarrow{\alpha} E}{\vdash_R \ AE \twoheadrightarrow AF \xrightarrow{\alpha} E}\ A \cap R = \emptyset,$$

expressing the idea that if no action ready in the environment can be accepted by the first component of a prisum then the capabilities of the second component can be realised. This does not work as desired. It allows, e.g., $\vdash_{\{a\}} \tau.\mathbf{0} \twoheadrightarrow a.0 \xrightarrow{a} \mathbf{0}$, whereas we want only a τ action to be possible. (Here a τ action plays a role analogous to that of a *SKIP* guard in a PRIALT construction in Occam.) To prevent the rule $\mathbf{Pri}_b$ from being applied, and so violating the pre-emptive power we wish to ascribe to τ actions in a prisum, we arrange that $\tau.\mathbf{0}\ \mathbf{ac}\ \{*\}$ via $\mathbf{ac}(ii)$ and that the side condition is replaced by $A \cap (R \cup \{*\}) = \emptyset$ in rule $\mathbf{Pri}_b$ in Fig. 4.

$\mathbf{ac}(i)$ $\mathbf{0}$ ac $\emptyset$ $\qquad$ $\mathbf{ac}(ii)$ $\tau.E$ ac $\{*\}$

$\mathbf{ac}(iii)$ $a.E$ ac $\{\overline{a}\}$

$$\mathbf{ac}(iv)\ \frac{G\ \mathbf{ac}\ A_0 \quad H\ \mathbf{ac}\ A_1}{G + H\ \mathbf{ac}\ A_0 \cup A_1} \qquad \mathbf{ac}(v)\ \frac{G\ \mathbf{ac}\ A}{G \setminus L\ \mathbf{ac}\ A - \overline{L}}\ (\text{Recall we assume } L \subseteq \mathcal{A})$$

$$\mathbf{ac}(vi)\ \frac{G\ \mathbf{ac}\ A}{G[f]\ \mathbf{ac}\ f(A)} \qquad \mathbf{ac}(vii)\ \frac{G[fix(X = G)/X]\ \mathbf{ac}\ A}{fix(X = G)\ \mathbf{ac}\ A}$$

$$\mathbf{ac}(viii)\ \frac{AE\ \mathbf{ac}\ A_0}{AE \twoheadrightarrow AF\ \mathbf{ac}\ A_0}\ (* \in A_0)$$

$$\mathbf{ac}(ix)\ \frac{AE\ \mathbf{ac}\ A_0 \quad AF\ \mathbf{ac}\ A_1}{AE \twoheadrightarrow AF\ \mathbf{ac}\ A_0 \cup A_1}\ (* \notin A_0)$$

FIG. 3. The definition of the acceptance function **ac**.

$$\mathbf{Pre}_a\ \vdash_R a.E \xrightarrow{a} E \text{ if } \overline{a} \in R$$

$\mathbf{Pre}_b$ $\vdash_R \overline{a}.E \xrightarrow{\overline{a}} E$ $\qquad$ $\mathbf{Pre}_c$ $\vdash_R \tau.E \xrightarrow{\tau} E$

$$\mathbf{Sum}_a\ \frac{\vdash_R E \xrightarrow{a} E'}{\vdash_R E + F \xrightarrow{a} E'} \qquad \mathbf{Sum}_b\ \frac{\vdash_R F \xrightarrow{a} F'}{\vdash_R E + F \xrightarrow{a} F'}$$

$$\mathbf{Res}\ \frac{\vdash_{R-\overline{L}} E \xrightarrow{a} E'}{\vdash_R E \setminus L \xrightarrow{a} E' \setminus L}\ (a \notin \overline{L}) \qquad \mathbf{Rel}\ \frac{\vdash_R E \xrightarrow{a} E'}{\vdash_{f(R)} E[f] \xrightarrow{f(a)} E'[f]}$$

(Recall we assume $L \subseteq \mathcal{A}$)

$$\mathbf{Pri}_a\ \frac{\vdash_R AE \xrightarrow{a} E}{\vdash_R AE \twoheadrightarrow AF \xrightarrow{a} E} \qquad \mathbf{Pri}_b\ \frac{AE\ \mathbf{ac}\ A \quad \vdash_R AF \xrightarrow{a} E}{\vdash_R AE \twoheadrightarrow AF \xrightarrow{a} E}\ A \cap (R \cup \{*\}) = \emptyset$$

$$\mathbf{Com}_a\ \frac{\vdash_{R \cup R_1} E \xrightarrow{a} E' \quad F\ \mathbf{re}\ R_1}{\vdash_R E|F \xrightarrow{a} E'|F}\ (a \in \mathcal{A} \Rightarrow \overline{a} \in R)$$

$$\mathbf{Com}_b\ \frac{\vdash_{R \cup R_0} F \xrightarrow{a} F' \quad E\ \mathbf{re}\ R_0}{\vdash_R E|F \xrightarrow{a} E|F'}\ (a \in \mathcal{A} \Rightarrow \overline{a} \in R)$$

$$\mathbf{Com}_c\ \frac{\vdash_{R \cup R_1} E \xrightarrow{a} E' \quad E\ \mathbf{re}\ R_0 \quad \vdash_{R \cup R_0} F \xrightarrow{\overline{a}} F' \quad F\ \mathbf{re}\ R_1}{\vdash_R E|F \xrightarrow{\tau} E'|F'}$$

$$\mathbf{Rec}\ \frac{\vdash_R E[fix(X = E)/X] \xrightarrow{a} E'}{\vdash_R fix(X = E) \xrightarrow{a} E'}$$

FIG. 4. The definition of the transition relation.

Consider the rule $\mathbf{Com}_c$ for compositions under the assumptions that $E\ \mathbf{re}\ R_0$ and $F\ \mathbf{re}\ R_1$. The rule takes account of the fact that the assumption that the environment of $E|F$ is ready to do R amounts to the assumption on E that its environment is ready with $R \cup R_1$, and similarly that F's environment is ready with $R \cup R_0$. If on these assumptions E and F can perform complementary actions then their composition can synchronise. The rule $\mathbf{Com}_c$ has a pleasing symmetry, though note that by virtue of Proposition 3.1 the requirements of $\mathbf{Com}_c$, $\mathbf{Com}_a$, and $\mathbf{Com}_b$ can be relaxed when they involve o-actions. Consequently these rules could be replaced by two rules taking account of the fact that transitions associated with o-actions do not depend on ready sets of the environment.

PROPOSITION 3.1. *For terms E, E' and all R, $S \subseteq \mathcal{A}$, $\vdash_R E \xrightarrow{\overline{a}} E' \Leftrightarrow \vdash_S E \xrightarrow{\overline{a}} E'$.*

PROPOSITION 3.2. *The relations* **ac** *and* **re** *are functions in the sense that AE* **ac** *A and AE* **ac** *A' implies $A = A'$ and E* **re** *R and E* **re** *R' implies $R = R'$. Moreover*

- E **re** $S \Leftrightarrow S = \{\bar{a} \in \bar{\mathscr{A}} \mid \forall R\, \exists E'.\ \vdash_R E \xrightarrow{\bar{a}} E'\}$,
- AE **ac** $A \Leftrightarrow A = \{\bar{\alpha} \in \bar{\mathscr{A}} \cup \{*\} \mid \exists R\, \exists E'.\ \vdash_R AE \xrightarrow{\alpha} E'\}$.

Notation. In future we will use $\mathbf{ac}(G)$ for the acceptance set of G and $\mathbf{re}(E)$ for the ready set of E—justified because the acceptance and ready relations are functions.

CCS^{+} essentially extends CCS in the sense that once we restrict to $+\!\!\!\triangleright$-free terms (those not mentioning $+\!\!\!\triangleright$), the transition relations agree but for the extra decoration of a ready set on the relations:

PROPOSITION 3.3. *For $+\!\!\!\triangleright$-free terms, $E \xrightarrow{\alpha} E'$ in* CCS *iff*

$$\vdash_R E \xrightarrow{\alpha} E' \text{ in CCS}^{+} \text{ for all } R \subseteq \bar{\mathscr{A}} \text{ such that } \alpha \in \mathscr{A} \Rightarrow \bar{\alpha} \in R.$$

4. STRONG BISIMULATION

We take a generalisation of Milner's strong bisimulation as our central equivalence between agents.

DEFINITION 4.1. A relation $\mathscr{Q} \subseteq \mathscr{P}^{+} \times \mathscr{P}^{+}$ is a strong bisimulation, with respect to the operational semantics, if $(P, Q) \in \mathscr{Q}$ implies, for all $\alpha \in Act$ and for all $R \subseteq \bar{\mathscr{A}}$,

(1) whenever $\vdash_R P \xrightarrow{\alpha} P'$ then, for some Q', $\vdash_R Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathscr{Q}$;

(2) whenever $\vdash_R Q \xrightarrow{\alpha} Q'$ then, for some P', $\vdash_R P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathscr{Q}$.

Let the relation $\sim$ on agents be defined by

$$P \sim Q \text{ iff } (P, Q) \in \mathscr{Q} \text{ for some strong bisimulation } \mathscr{Q}.$$

As emphasised by Park it follows by basic fixed-point theory that $\sim$ is an equivalence relation and the largest strong bisimulation, and as such it is amenable to the proof technique: *to show $P \sim Q$ it is sufficient to exhibit a strong bisimulation containing (P, Q).*

In particular, this technique is used to show basic properties of operations such as their associativity and commutativity. For example, to show that $(P \mid Q) \mid R \sim P \mid (Q \mid R)$, in general, it is sufficient to show that the relation consisting of all pairs $((P \mid Q) \mid R, P \mid (Q \mid R))$ is a strong bisimulation, a routine but lengthy process.

PROPOSITION 4.1.

$$(P \mid Q) \mid R \sim P \mid (Q \mid R) \quad \text{and} \quad P \mid Q \sim Q \mid P$$
$$(P + Q) + R \sim P + (Q + R) \quad \text{and} \quad P + Q \sim Q + P$$
$$(P +\!\!\!\triangleright Q) +\!\!\!\triangleright R \sim P +\!\!\!\triangleright (Q +\!\!\!\triangleright R).$$

This technique is used to show that equivalence is a congruence; i.e., it is substitutive under the constructs of CCS^{+} as well as under recursive definition. The proof follows the standard lines of [16], though with this more intricate operational semantics, it is necessary to check that the strong equivalence of two terms implies that they have the same acceptance and ready sets (this is needed in proving the congruence of $\sim$ with respect to $+\!\!\!\triangleright$ and $\mid$):

LEMMA 4.2. *If $P \sim Q$ then* $\mathbf{ac}(P) = \mathbf{ac}(Q)$ *and* $\mathbf{re}(P) = \mathbf{re}(Q)$.

PROPOSITION 4.3. *Let $P_1 \sim P_2$; then*

(1) $\alpha.P_1 \sim \alpha.P_2$
(2) $P_1 + Q \sim P_2 + Q$
(3) $P_1 +\!\!\!\triangleright Q \sim P_2 +\!\!\!\triangleright Q$
(4) $Q +\!\!\!\triangleright P_1 \sim Q +\!\!\!\triangleright P_2$
(5) $P_1 \mid Q \sim P_2 \mid Q$
(6) $P_1 \backslash L \sim P_2 \backslash L$
(7) $P_1[f] \sim P_2[f]$.

Thus strong equivalence is preserved by prefixing, sum, prisum, composition, restriction, and relabelling. We remark that due to the syntactic restriction placed on operands of prisum, P_1, P_2, and Q in statements (3) and (4) must be terms in $\mathscr{A}\mathscr{E}^{+}$, otherwise $P_1 +\!\!\!\triangleright Q$, $Q +\!\!\!\triangleright P_2$, etc. would not be terms in $\mathscr{E}^{+}$. For example, let $E \equiv (\bar{a}.\mathbf{0} + b.\mathbf{0}) \backslash a$ and $F \equiv b.\mathbf{0}$—i.e., suppose that F is an element of $\mathscr{A}\mathscr{E}^{+}$ and E is not. Although $E \sim F$, $E +\!\!\!\triangleright AE \not\sim F +\!\!\!\triangleright AE$ for $AE \in \mathscr{A}\mathscr{E}^{+}$.

Hitherto, strong equivalence has been defined only for processes—i.e., closed expressions in $\mathscr{E}^{+}$. To remedy this, we extend the definition of $\sim$ to open terms as follows. Let a *substitution* σ map term variables to closed terms such that $E\sigma$ represents the closed term resulting from the substitution of all free variables X in E by $\sigma(X)$.

DEFINITION 4.2. Let E and F be terms, possibly with free variables. Then $E \sim F$ if $E\sigma \sim F\sigma$ for any substitution σ with domain containing at least the free variables in E and F.

It is a simple matter to extend Proposition 4.3 to open terms. Proposition 4.4 shows, moreover, that recursion preserves strong equivalence.

PROPOSITION 4.4. *If $E \sim F$ then $fix(X = E) \sim fix(X = F)$.*

THEOREM 4.5. *$\sim$ is a congruence with respect to the operators of* CCS^{+}, *subject to the syntactic restrictions on operands of prisum.*

Finally, we note that the extension of CCS to priority does not lead to any new identifications between closed terms of CCS:

PROPOSITION 4.6. *For closed $\Rightarrow$-free terms, E and E' are strongly equivalent in the sense of Milner iff $E \sim E'$ in* $\mathrm{CCS}^{\Rightarrow}$.

This will mean that the equational laws of the next section are conservative over CCS terms: closed terms of CCS will only be provably equal iff they formerly were so in CCS.

5. EQUATIONAL LAWS

We now present a set of equational laws which are complete with respect to finite $\mathrm{CCS}^{\Rightarrow}$ terms (a term is *finite* if it contains no variables). First, the usual rules of equational reasoning (reflexivity, symmetry, transitivity, and Leibnitz' rule, viz., "substitution of equals for equals") hold, as $\sim$ is a congruence. Further rules are presented in Fig. 5. These consist of Milner's laws for strong equivalence together with new laws for the prisum operator, notably **P1–P9**. For instance, **P4** shows how prisum distributes over sum on the right-hand side (but not on the left; this would be unsound in general). The laws **P5** and **P7** are similar to each other; they both express the priority of actions on the left of a prisum. The laws of Fig. 5 with the addition of the expansion law (Fig. 6) are complete for finite terms. The expansion law operates on terms in *normal* form. The notation and this concept need some explanation.

Notation. We shall form sums and prisums over finite total orders.

A term $(G_{i_0} \Rightarrow (\cdots \Rightarrow (G_{i_k} \Rightarrow \cdots)\cdots))$ is written as $G_{i_0} \Rightarrow \cdots \Rightarrow G_{i_k} \Rightarrow \cdots$ or $\Rrightarrow \sum_{i \in I} G_i$, where I is the finite ordered set $i_0 < \cdots i_k < \ldots$.

A1 $P + Q = Q + P$ **A3** $P + (Q + R) = (P + Q) + R$

A2 $P + P = P$ **A4** $P + 0 = P$

P1 $\mathcal{AE} \Rightarrow \mathcal{AE} = \mathcal{AE}$ **P4** $(\mathcal{AE} \Rightarrow G) + (\mathcal{AE} \Rightarrow H) = \mathcal{AE} \Rightarrow (G + H)$

P2 $\mathcal{AE} \Rightarrow 0 = \mathcal{AE}$ **P5** $(\tau.E + G) \Rightarrow \mathcal{AE} = \tau.E + G$

P3 $0 \Rightarrow \mathcal{AE} = \mathcal{AE}$ **P6** $(\mathcal{AE}_0 \Rightarrow \mathcal{AE}_1) \Rightarrow \mathcal{AE}_2 = \mathcal{AE}_0 \Rightarrow (\mathcal{AE}_1 \Rightarrow \mathcal{AE}_2)$

P7 $(G + \alpha.E_0) \Rightarrow \mathcal{AE} \Rightarrow \alpha.E_1 = (G + \alpha.E_0) \Rightarrow \mathcal{AE}$

P8 $\mathcal{AE}_0 + (\mathcal{AE}_1 \Rightarrow \mathcal{AE}_0) = \mathcal{AE}_0 + \mathcal{AE}_1$

P9 $\mathcal{AE}_0 \Rightarrow \mathcal{AE}_1 = \mathcal{AE}_0 + (\mathcal{AE}_0 \Rightarrow \mathcal{AE}_1)$

C1 $P|Q = Q|P$ **C3** $P|(Q|R) = (P|Q)|R$

C2 $P|0 = P$

R1 $0[f] = 0$ **R3** $(\alpha.P)[f] = f(\alpha).(P[f])$

R2 $(P + Q)[f] = P[f] + Q[f]$ **R4** $(\mathcal{AE}_0 \Rightarrow \mathcal{AE}_1)[f] = \mathcal{AE}_0[f] \Rightarrow \mathcal{AE}_1[f]$

L1 $0 \setminus L = 0$

L2 $(\alpha.P) \setminus L = \begin{cases} 0 & \text{if } \alpha \in L \cup \bar{L} \\ \alpha.(P \setminus L) & \text{otherwise} \end{cases}$

L3 $(P + Q) \setminus L = (P \setminus L) + (Q \setminus L)$

L4 $(\mathcal{AE}_0 \Rightarrow \mathcal{AE}_1) \setminus L = (\mathcal{AE}_0 \setminus L) \Rightarrow (\mathcal{AE}_1 \setminus L)$

Rec1 $fix(X = E) = E[fix(X = E)/X]$

Rec2 $F = fix(X = E)$ if $F = E[F/X]$

FIG. 5. Equational laws satisfied by $\sim$.

Suppose $P \equiv \sum_{i \in I} \Rrightarrow \sum_{j \in J_i} \sum_{k \in K_{ij}} \alpha^j_{ik}.P^j_{ik}$ and $Q \equiv \sum_{l \in L} \Rrightarrow \sum_{m \in M_l} \sum_{n \in N_{lm}} \beta^m_{ln}.Q^m_{ln}$ then

$$P|Q = \sum_{i \in I} \Rrightarrow \sum_{j \in J_i} \left(\sum_{k \in K_{ij}} \alpha^j_{ik}.(P^j_{ik}|Q) + \sum_{\overline{\alpha^j_{ik}} = \beta^m_{ln} \in \overline{\mathcal{A}}} \tau.(P^j_{ik}|Q^m_{ln}) \right) +$$

$$\sum_{i \in L} \Rrightarrow \sum_{m \in M_l} \left(\sum_{n \in N_{lm}} \beta^m_{ln}.(P|Q^m_{ln}) + \sum_{\overline{\beta^m_{ln}} = \alpha^j_{ik} \in \overline{\mathcal{A}}} \tau.(P^j_{ik}|Q^m_{ln}) \right)$$

FIG. 6. The expansion law.

Similarly, a term $(G_{i_0} + (\cdots + (G_{i_k} + \cdots)\cdots))$ is written as $G_{i_0} + \cdots + G_{i_k} + \cdots$ or $\sum_{i \in I} G_i$, where I is the finite ordered set $i_0 < \cdots i_k < \ldots$. Because $+$ is commutative and associative with respect to $\sim$ we do not care about the precise order on the indexing sets of sums—we just need the order to write down the sum as an expression. Conceptually it is simplest to imagine every finite indexing set having been given a total order once and for all. Then we can allow sums $\sum_{i \in I} G_i$ indexed by sets I with their ordering understood as having been settled. Furthermore, we shall then understand $\sum_{P(G_i)} G_i$, where P is a predicate, as an abbreviation for the sum $\sum_{i \in I'} G_i$ indexed by $I' = \{i \in I \mid P(G_i)\}$. To take a somewhat complicated example, the component sum

$$\sum_{\beta^m_{ln} = \alpha^j_{ik} \in \mathcal{A}} \tau.(P^j_{ik} \mid Q^m_{ln}),$$

seen in the expansion law, is the sum of all terms $\tau.(P^j_{ik} \mid Q^m_{ln})$ indexed by the set of (i, j, k, n) for which $i \in I, j \in J_i, k \in K_{ij}$, $n \in N_{lm}$ such that $\bar{\beta}^m_{ln} = \alpha^j_{ik} \in \mathcal{A}$.

The presence of prisums in the syntax complicates the normal form. By analogy with CCS we expect that a closed expression G of $\mathcal{G}^{\Rightarrow}$ should be equivalent to a sum of prefixes, a *sum form*. This would make a closed prisum expression AE in $\mathcal{A}\mathcal{E}^{\Rightarrow}$ equivalent to an indexed prisum of sum forms, a *prisum form*. We cannot expect better than a sum of prisum forms as a *normal form* for expression E in $\mathcal{E}^{\Rightarrow}$, and it is with such forms the expansion law deals.

DEFINITION 5.1. A term is said to be in *normal form*:

$$\sum_{i \in I} \Rrightarrow \sum_{j \in J_i} \sum_{k \in K_{ij}} \alpha^j_{ik}.P^j_{ik} \text{ if for all } i \in I,\ j \in J_i,$$
$$k \in K_{ij},\ P^j_{ik} \text{ are in normal form.} \qquad (1)$$

The expansion law shows how the parallel composition of normal forms yields a normal form. In the case of $\Rightarrow$-free expressions, where the forms have at most unary prisums, the expansion law reduces to the one familiar from CCS. The proof of soundness of the expansion law is nontrivial and appears in the Appendix.

THEOREM 5.1 (Soundness). *The equational laws of Fig. 5 are valid (i.e., for each law $P = Q$ it is true that $P \sim Q$). The*

expansion law of Fig. 6 is sound (in the sense of preserving valid equations). Any equation deduced by equational reasoning from these laws is valid. (See the appendix for the proof of soundness of the expansion law.)

The proof of completeness follows the lines of first converting terms into a normal form and then proving the normal forms equal if they are strongly equivalent. The completeness proof uses a more refined notion of normal form:

DEFINITION 5.2. Say that a term P, in the normal form $\sum_{i\in I}\sum_{j\in J_i}\sum_{k\in K_{ij}}\alpha_{ik}^{j}.P_{ik}^{j}$, is in *strong normal form* if

- $\alpha_{ik}^{j}=\tau$ implies that j is the maximum element of J_i,
- $\alpha_{ik}^{j}=\alpha_{ik}^{j'}$ implies that $j=j'$.

The importance of a strong normal form, such as that above, is that any prefix component $\alpha_{ik}^{j}.P_{ik}^{j}$ is associated with transitions that the normal form can make. This means that we can match to within $\sim$ prefix components of two $\sim$-equivalent strong normal forms.

PROPOSITION 5.2. *For P in strong normal form as above, for any $i\in I, j_0\in J_i, k_0\in K_{ij_0}$ for which $\alpha_{ik_0}^{j_0}\in\mathscr{A}$ we have*

$$\vdash_R P\xrightarrow{\alpha_{ik_0}^{j_0}}P_{ik_0}^{j_0}$$

if $R\cap\{\bar{\alpha}_{ik}^{j}\mid j<j_0\ \&\ k\in K_{ij}\}=\varnothing$ and $\bar{\alpha}_{ik_0}^{j_0}\in R$.

PROPOSITION 5.3. *For every closed term P there is a term P' in strong normal form such that $P=P'$.*

Proof. The equational laws enable any finite term to be converted to normal form. By application of **P5**, **P7**, **P4**, **A1**, and **A3** any normal form can be converted to one of strong normal form. ▮

Once two strongly equivalent terms are in strong normal form they can be proved equal by showing using equational reasoning that any prisum component of one can be absorbed[2] by the other. Achieving this involves some equational "surgery" on prisum components. Its need is seen in the strong equivalence

$$(a.0+b.0)\pitchfork c.0\sim(a.0\pitchfork b.0\pitchfork c.0)+(b.0\pitchfork a.0\pitchfork c.0)$$

between two rather different strong normal forms. The surgery is managed with the help of two central lemmas, the "topping" and the "splitting" lemma.

LEMMA 5.4. *If $\mathbf{ac}(AE_1)\subseteq\mathbf{ac}(AE_0)$ then $AE_0\pitchfork AE_1=AE_0$.*

[2] We say that P absorbs Q if $P+Q=P$.

Proof. Without loss of generality we can assume that AE_0 and AE_1 are in normal form. The result then follows by repeated use of ***P5***, ***P4***, ***P7***, **A1**, and **A3**, and finally **A2** to eliminate prefixed terms in AE_1. ▮

LEMMA 5.5 (Topping Lemma). *If $\mathbf{ac}(AE_1)\subseteq\mathbf{ac}(AE_0)$ then $AE_0+(AE_1\pitchfork AE_2)=(AE_0\pitchfork AE_2)+(AE_1\pitchfork AE_2)$.*

Proof. By Lemma 5.4 we obtain

$$AE_0\pitchfork AE_1=AE_0. \tag{2}$$

Now we reason:

$$\begin{aligned}
&AE_0+(AE_1\pitchfork AE_2)\\
&\quad=(AE_1\pitchfork AE_2)+AE_0\\
&\qquad\text{by A1}\\
&\quad=(AE_1\pitchfork AE_2)+((AE_0\pitchfork AE_1)\pitchfork AE_2)\\
&\qquad\text{by P8 and P6}\\
&\quad=(AE_1\pitchfork AE_2)+(AE_0\pitchfork AE_2)\\
&\qquad\text{by (2).}\ \blacksquare
\end{aligned}$$

We remark that law **P8** can be regarded as a special case of the topping lemma because

$$\begin{aligned}
AE_0+AE_1&=AE_0+(\mathbf{0}\pitchfork AE_1)\\
&\qquad\text{by P3}\\
&=(AE_0\pitchfork AE_1)+(\mathbf{0}\pitchfork AE_1)\\
&\qquad\text{by "topping" as } \mathbf{ac}(\mathbf{0})=\varnothing\\
&=(AE_0\pitchfork AE_1)+AE_1\\
&\qquad\text{by P3.}
\end{aligned}$$

LEMMA 5.6 (Splitting Lemma).

$$\begin{aligned}
&AE_0\pitchfork(G+H)\pitchfork AE_1\\
&\quad=(AE_0\pitchfork G)+(AE_0\pitchfork(G+H)\pitchfork AE_1).
\end{aligned}$$

Proof.

$$\begin{aligned}
&AE_0\pitchfork(G+H)\pitchfork AE_1\\
&\quad=(AE_0\pitchfork(G+H))\\
&\qquad+(AE_0\pitchfork(G+H)\pitchfork AE_1)\\
&\qquad\text{by P9}\\
&\quad=(AE_0\pitchfork G)+(AE_0\pitchfork H)\\
&\qquad+(AE_0\pitchfork(G+H)\pitchfork AE_1)\\
&\qquad\text{by P4}
\end{aligned}$$

$$= (AE_0 \twoheadrightarrow G) + (AE_0 \twoheadrightarrow G) + (AE_0 \twoheadrightarrow H) + (AE_0 \twoheadrightarrow (G + H) \twoheadrightarrow AE_1)$$

by **A2**

$$= (AE_0 \twoheadrightarrow G) + (AE_0 \twoheadrightarrow (G + H) \twoheadrightarrow AE_1)$$

by **P4** and **P9**. ∎

By "topping" and "splitting" we prove completeness of the equational proof system for finite terms. The proof is given in the Appendix.

THEOREM 5.7 (Completeness). *Let P, Q be finite terms; then $P \sim Q$ iff $P = Q$.*

An idea of the role played by the "topping" and "splitting" lemmas is illustrated in showing that $P = Q$, where

$$P \equiv (a.0 + b.0) \twoheadrightarrow c.0$$

and

$$Q \equiv (a.0 \twoheadrightarrow b.0 \twoheadrightarrow c.0) + (b.0 \twoheadrightarrow a.0 \twoheadrightarrow c.0).$$

To establish this on the lines followed in the completeness proof we first argue that

$$Q = (a.\mathbf{0} + b.\mathbf{0}) + (a.0 \twoheadrightarrow b.\mathbf{0} \twoheadrightarrow c.\mathbf{0}) + (b.\mathbf{0} \twoheadrightarrow a.0 \twoheadrightarrow c.\mathbf{0})$$

by "splitting" twice

$$= ((a.\mathbf{0} + b.\mathbf{0}) \twoheadrightarrow c.\mathbf{0}) + (a.0 \twoheadrightarrow b.\mathbf{0} \twoheadrightarrow c.\mathbf{0}) + (b.\mathbf{0} \twoheadrightarrow a.0 \twoheadrightarrow c.\mathbf{0})$$

by "topping."

Thus Q can absorb the prisum component $((a.\mathbf{0} + b.\mathbf{0}) \twoheadrightarrow c.\mathbf{0})$. Similarly

$$P = a.\mathbf{0} + b.\mathbf{0} + ((a.\mathbf{0} + b.\mathbf{0}) \twoheadrightarrow c.\mathbf{0})$$

by "splitting" twice

$$= (a.\mathbf{0} \twoheadrightarrow b.\mathbf{0}) + b\mathbf{0} + ((a.\mathbf{0} + b.\mathbf{0}) \twoheadrightarrow c.\mathbf{0})$$

bt **P8**—as remarked, a form of "topping"

$$= (a.\mathbf{0} \twoheadrightarrow b.\mathbf{0}) + ((a.\mathbf{0} + b.\mathbf{0}) \twoheadrightarrow c.\mathbf{0})$$

by "splitting" in reverse

$$= (a.\mathbf{0} \twoheadrightarrow b.\mathbf{0} \twoheadrightarrow c.\mathbf{0}) + ((a.\mathbf{0} + b.\mathbf{0}) \twoheadrightarrow c.\mathbf{0})$$

by "topping."

Thus P can absorb the prisum component $(a.\mathbf{0} \twoheadrightarrow b.\mathbf{0} \twoheadrightarrow c.\mathbf{0})$ and similarly the component $(b.\mathbf{0} \twoheadrightarrow a.\mathbf{0} \twoheadrightarrow c.\mathbf{0})$. It follows that $P = P + Q = Q$. This is certainly not the most direct proof, nor was it meant to be. However, it does illustrate how the splitting and topping lemmas can be used systematically to prove with the laws the equivalence of two strongly normal forms.

6. EXAMPLES

6.1. *The Tick-Tock Example*

We return to the "tick-tock" example. Consider the application of the laws on SYS'—i.e., the state of the system after a *shut-down* action occurs:[3]

$$SYS' = (C \mid S')\backslash i$$
$$S' = \bar{i}.\mathbf{0}$$
$$C = i.\mathbf{0} \twoheadrightarrow (tick.C + tock.C).$$

Applying the expansion law to SYS' yields

$$(\bar{i}.(C \mid \mathbf{0}) + (((i.(\mathbf{0} \mid S') + \tau.(\mathbf{0} \mid \mathbf{0})) \twoheadrightarrow (tick.(C \mid S') + tock.(C \mid S'))))\backslash i.$$

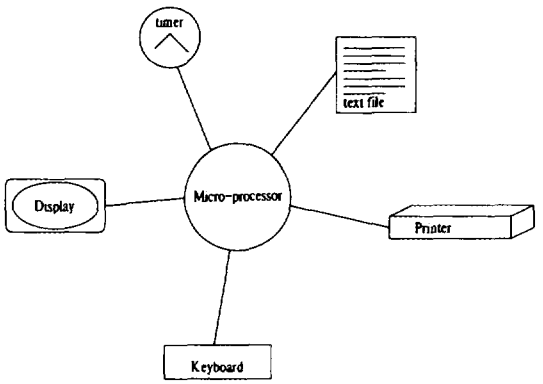
Applying **L1**–**L3**, **C2**, and, optionally, either of **P5** or **P2**, **L4** to remove the prisum yields $\tau.\mathbf{0}$, as expected.

6.2. *A Toy Distributed System*

This example is an adaptation of one found in [2]. Consider a distributed system comprising a visual display unit, a timer, a data file, a printer, a keyboard, and a microprocessor which coordinates the printing of the data file, the display of the printer status and the servicing of interrupts from the timer and the keyboard (see Fig. 7).

The system is either active (i.e., printing) or inactive (i.e., not printing). We assume that initially it is inactive. In the initial state, the timer periodically prompts the microprocessor to display diagnostic information. The system remains inactive until a user types *Start* on the keyboard. This done, the microprocessor gives priority to printing the file over the display of diagnostics. The file sends words, character by character, to the microprocessor, which then forwards them to the printer. The user can type *stop* at any time. This instructs the microprocessor to deactivate printing and reinstate the system to the inactive state. Figure 7 gives a description of this system in CCS augmented with the priority operator. The operator is used to enforce the "immediate" response to interrupts.

[3] Strictly speaking, to fit this description into our syntax involves taking C to be $fix(X = (i.\mathbf{0} \twoheadrightarrow (tick.X + tock.X))$.



$$
\begin{aligned}
Timer &= \overline{status}.Timer \\
MP_I &= sp.\overline{ps}.\overline{fs}.\overline{started}.MP_A + status.\overline{ss}.sa.MP_I \\
MP_A &= int.\overline{fh}.\overline{ph}.\overline{stopped}.MP_I + char.\overline{pchar}.MP_A \\
Display &= ss.\overline{diag}.\overline{sa}.Display \\
File &= fs.Send \\
Send &= \overline{char}.Send + fh.File \\
Printer &= ps.Print \\
Print &= ph.Printer + pchar.Print \\
Keyboard &= start.\overline{sp}.started.Keyboard_0 \\
Keyboard_0 &= stop.\overline{int}.stopped.Keyboard \\
Sys &= \langle Timer \mid MP_I \mid Display \mid File \mid Printer \mid Keyboard \rangle \setminus I \\
\text{where } I &= \{status, started, stopped, char, pchar, fs, ps, fh, ph, sp, int, ss, sa\}
\end{aligned}
$$

FIG. 7. A toy distributed system.

Consider the initial state

$$Sys = \langle Timer \mid MP_I \mid Display \mid File \mid Printer \mid Keyboard \rangle \setminus I,$$

where I is the set of hidden actions defined in Fig. 7. In any environment (i.e., for any ready-set $R \subseteq \mathscr{A}$), the timer can periodically prompts the microprocessor to display the status of the printer:

$$\xrightarrow[R]{\tau} \langle Timer \mid \overline{ss}.sa.MP_I \mid Display \mid File \mid Printer \mid Keyboard \rangle \setminus I$$

$$\xrightarrow[R]{\tau} \langle Timer \mid sa.MP_I \mid \overline{diag}.\overline{sa}.Display \mid File \mid Printer \mid Keyboard \rangle \setminus I$$

$$\xrightarrow[R]{\overline{diag}} \langle Timer \mid sa.MP_I \mid \overline{sa}.Display \mid File \mid Printer \mid Keyboard \rangle \setminus I$$

$$\xrightarrow[R]{\tau} Sys.$$

A user can type *start* any time while the system is inactive. After *start* is typed, the microprocessor handles any outstanding interactions (if any) with the timer and display, and proceeds to the state

$$Sys_0 = \langle Timer \mid MP_I \mid Display \mid File \mid Printer \mid \overline{sp}.started.Keyboard_0 \rangle \setminus I.$$

The behaviour of the system in state Sys can be specified as

$$\begin{aligned} Spec = {} & start.Spec_0 + \tau.(start.\tau.\overline{diag}.\tau.Spec_0 \\ & + \tau.(\overline{diag}.(\tau.Spec + start.\tau.Spec_0) \\ & + start.\overline{diag}.\tau.Spec_0)), \end{aligned}$$

where $Spec_0$ specifies the behaviour of the system in state Sys_0 as described below. When the system is in state Sys_0, since the keyboard is willing to perform $\overline{sp}$, the priority operator in MP_I ensures that the system becomes active immediately (i.e., futher interrupts from the timer are precluded until the system is activated). In all environments (i.e., for all ready-sets R), the system proceeds as follows. First the microprocessor handshakes with the keyboard:

$$\xrightarrow[R]{\tau} \langle Timer \mid \overline{ps}.\overline{fs}.\overline{started}.MP_A \mid Display \mid File \mid Printer \mid started.Keyboard_0 \rangle \setminus I.$$

The microprocessor then sets the printer to an active state,

$$\xrightarrow[R]{\tau} \langle Timer \mid \overline{fs}.\overline{started}.MP_A \mid Display \mid File \mid Printer \mid started.Keyboard_0 \rangle \setminus I.$$

then initializes the character flow from the file,

$$\xrightarrow[R]{\tau} \langle Timer \mid \overline{started}.MP_A \mid Display \mid File \mid Printer \mid started.Keyboard_0 \rangle \setminus I.$$

and then acknowledges the keyboard,

$$\xrightarrow[R]{\tau} \langle Timer \mid MP_A \mid Display \mid Send \mid Print \mid Keyboard_0 \rangle \setminus I.$$

The system is now in an active state

$$Sys_1 = \langle Timer \mid MP_A \mid Display \mid Send \mid Print \mid Keyboard_0 \rangle \setminus I$$

from which character flow from file to printer via the microprocessor can commence. If $Spec_1$ represents the behaviour of the system in state Sys_1, then

$$Spec_0 = \tau.\tau.\tau.\tau.Spec_1,$$

as illustrated above by the transition path from state Sys_0 to Sys_1. In state Sys_1, a typical character transaction from file to printer in any environment (i.e., for all ready-sets R) is illustrated by the following transitions. First a character is passed from the file to the microprocessor:

$$\xrightarrow[R]{\tau} \langle Timer \mid \overline{pchar}.MP_A \mid Display \mid Send \mid Print \mid Keyboard_0 \rangle \backslash I.$$

The character is then forwarded to the printer and the system is restored to state Sys_1:

$$\xrightarrow[R]{\tau} \langle Timer \mid MP_A \mid Display \mid Send \mid Print \mid Keyboard_0 \rangle \backslash I.$$

While the system is active, the user can type *stop* at the keyboard at any time to interrupt the process of printing. Note that the use of the priority operator in MP_A ensures that this interrupt has overall priority in this circumstance. On interruption (i.e., when the user types *stop* at the keyboard), the microprocessor handles any outstanding interactions (if any) with the file and printer; then because the keyboard can perform $\overline{int}$ the active microprocessor in state MP_A proceeds immediately to deactivate the system. The behaviour of the system in state Sys_1 is specified by $Spec_1$, where

$$Spec_1 = stop.Spec_2 + \tau.(stop.\tau.Spec_2 + \tau.Spec_1),$$

in which $Spec_2 = \tau.\tau.\tau.\tau.Spec$.

This example, though unambitious, illustrated the use of the priority operator to guarantee the prompt servicing of interrupts. The microprocessor would be rather ineffective if it had to choose nondeterministically whether to service an interrupt or to continue with its "normal operation." More ambitious examples have been tackled with the help of the extended Concurrency Workbench [14] mentioned in the introduction. The augmented tools of the Workbench were used to check that *Sys* is strongly equivalent to *Spec*, as defined above.

7. CONCLUSION

We have recast the original work of Milner and Hennessy be developing the theory of strong bisimulation for an extension of CCS to include a priority choice operator, prisum. This increases the modelling power of CCS is an essential way. For example, it is now possible to define a fair scheduler. The agent $S1$ is given by the equations

$$S1 = a.S2 \twoheadrightarrow b.S1$$
$$S2 = b.S1 \twoheadrightarrow a.S2$$

defines a scheduler which alternates between giving preference to a and then to b actions. In the absence of any attempts to perform b actions it can repeatedly perform a's; otherwise it is guaranteed to perform a b action within two steps (see [15]). Future work, already begun by Torp Jensen, should look at how to extend the theory of weak bisimulation to cater for priority constructs. Present work [15] is providing an operational semantics which reflects the independence of events in a similar manner to that in Petri nets and event structures. The new structures of transition systems with independence (see [19]), which can be generated in much the same way as the operational semantics here, provide a semantics formally related to these more detailed, established models. Through these structures it is hoped to obtain a better analysis of locality in priority, as well as certain aspects of fairness. Attempts to simplify our work on priority, and an eye to broadening the range of application, are leading to more primitive operations with which to express priority (see [7]).

APPENDIX

LEMMA (Soundness of the expansion theorem). *The expansion theorem of Fig. 6 is sound.*

Proof. Suppose that

$$P \equiv \sum_{m \in M} \rangle\!\sum_{v \in V_m} \sum_{i \in I_{mv}} \alpha^v_{mi}.P^v_{mi}$$

and

$$Q \equiv \sum_{n \in N} \rangle\!\sum_{u \in U_n} \sum_{j \in J_{nu}} \beta^u_{nj}.Q^u_{nj}.$$

Assume that $\mathbf{re}(P) = R_0$ and $\mathbf{re}(Q) = R_1$, and abbreviate the right-hand side of the expansion law in Fig. 6 by *exp*. To establish that $P \mid Q \sim exp$ it is sufficient to show that

$$\vdash_R P \mid Q \xrightarrow{\mu} E \quad \Leftrightarrow \quad \vdash_R exp \xrightarrow{\mu} E$$

for all $R \subseteq \bar{\mathscr{A}}$, actions μ, and agents E.

($\Rightarrow$) There are several ways in which a transition $\vdash_R P \mid Q \xrightarrow{\mu} E$ can be derived; either by rules $\mathbf{Com}_a$,

Com$_b$, or **Com**$_c$. Consider only the latter case; the proof for the others is similar. In this case $\mu = \tau$ and

$$\vdash_{R \cup R_1} P \xrightarrow{\alpha} P' \text{ and } \vdash_{R \cup R_0} Q \xrightarrow{\bar{\alpha}} Q',$$
$$\text{such that } \mathbf{re}(P) = R_0 \text{ and } \mathbf{re}(Q) = R_1.$$

Assume that $\alpha = a$, say, in $\mathscr{A}$ (the argument when $\bar{\alpha} \in \mathscr{A}$ is symmetric). Then from the form of Q we must have $\bar{a} = \beta^{u_0}_{n_0 j_0}$ and $Q' \equiv Q^{u_0}_{n_0 j_0}$ for some $n_0 \in N$, $u_0 \in U_{n_0}$ and $j_0 \in J_{n_0 u_0}$. From the form of P, we see that $a = \alpha^{v_0}_{m_0 i_0}$ and $P' \equiv P^{v_0}_{m_0 i_0}$ for some $m_0 \in M_0$, $v_0 \in V_{m_0}$, and $i \in I_{m_0 v_0}$ such that

$$\vdash_{R \cup R_1} \sum_{v \in V_{m_0}} \sum_{i \in I_{m_0 v}} \alpha^v_{m_0 i}.P^v_{m_0 i} \xrightarrow{a} P^{v_0}_{m_0 i_0}.$$

The rules for prisum, sum, and prefixing (see Figs. 3–4) ensure that in order for this transition to be derivable the following must hold:

$$\{\bar{\alpha}^v_{m_0 i} \mid v < v_0 \wedge i \in I_{m_0 v}\} \cap (R \cup R_1 \cup \{*\}) = \varnothing.$$

This implies that

$$\{\bar{\alpha}^v_{m_0 i} \mid v < v_0 \wedge i \in I_{m_0 v}\} \cap R_1 = \varnothing \tag{1}$$

and

$$\{\bar{\alpha}^v_{m_0 i} \mid v < v_0 \wedge i \in I_{m_0 v}\} \cap (R \cup \{*\}) = \varnothing. \tag{2}$$

Consider now the prisum

$$\sum_{v \in V_{m_0}} \left(\sum_{i \in I_{m_0 v}} \alpha^v_{m_0 i}.(P^v_{m_0 i} \mid Q) + \sum_{\alpha^v_{m_0 i} = \beta^u_{nj} \in \mathscr{A}} \tau.(P^v_{m_0 i} \mid Q^u_{nj}) \right)$$

occurring in the expression *exp*. It contains the τ-prefixed term $\tau.(P^{v_0}_{m_0 i_0} \mid Q^{u_0}_{n_0 j_0})$; i.e., $\tau.(P' \mid Q')$ is in the v_0-indexed components of the prisum. Moreover, by (1), no τ-prefixed term $\tau.(P^v_{m_0 i} \mid Q^u_{nj})$ can appear in a v-component of the prisum for $v < v_0$. These facts and (2) yield

$$\vdash_R \sum_{v \in V_{m_0}} \left(\sum_{i \in I_{m_0 v}} \alpha^v_{m_0 i}.(P^v_{m_0 i} \mid Q) + \sum_{\alpha^v_{m_0 i} = \beta^u_{nj} \in \mathscr{A}} \tau.(P^v_{m_0 i} \mid Q^u_{nj}) \right) \xrightarrow{\tau} (P' \mid Q').$$

Consequently, $\vdash_R exp \xrightarrow{\tau} (P' \mid Q')$—i.e., the original transition of $(P \mid Q)$ is matched by one of *exp*.

($\Rightarrow$) One also requires that any transition of *exp* can be matched by one of $P \mid Q$. For example, one transition is

$$\vdash_R exp \xrightarrow{\tau} (P^{v_0}_{m_0 i_0} \mid Q^{u_0}_{n_0 j_0}),$$

where $\bar{\alpha}^{v_0}_{m_0 i_0} = \beta^{u_0}_{n_0 j_0} = \bar{a}$ say in $\bar{\mathscr{A}}$. For this transition to be derivable one must necessarily have that

$$\vdash_R \sum_{v \in V_{m_0}} \left(\sum_{i \in I_{m_0 v}} \alpha^v_{m_0 i}.(P^v_{m_0 i} \mid Q) + \sum_{\alpha^v_{m_0 i} = \beta^u_{nj} \in \mathscr{A}} \tau.(P^v_{m_0 i} \mid Q^u_{nj}) \right) \xrightarrow{\tau} (P^{v_0}_{m_0 i_0} \mid Q^{u_0}_{n_0 j_0}).$$

However, from the rules of prisum this is only possible if the acceptance sets of all v-indexed components of this prisum for $v < v_0$ are not contained in R. This implies that

$$\{\bar{\alpha}^v_{m_0 i} \mid v < v_0 \wedge i \in I_{m_0 v}\} \cap R = \varnothing \tag{3}$$

and further that no τ-prefixed terms (which accept $*$) occur in v-indexed components for $v < v_0$. Recalling that Q **re** R_1, the latter yields

$$\{\bar{\alpha}^v_{m_0 i} \mid v < v_0 \wedge i \in I_{m_0 v}\} \cap R_1 = \varnothing. \tag{4}$$

Combining (3) and (4), one sees that

$$\{\bar{\alpha}^v_{m_0 i} \mid v < v_0 \wedge i \in I_{m_0 v}\} \cap (R \cup R_1 \cup \{*\}) = \varnothing.$$

It now follows from the prisum rules that

$$\vdash_{R \cup R_1} \sum_{v \in V_{m_0}} \sum_{i \in I_{m_0 v}} \alpha^v_{m_0 i}.P^v_{m_0 i} \xrightarrow{a} P^{v_0}_{m_0 i_0}.$$

Hence $\vdash_{R \cup R_1} P \xrightarrow{a} P^{v_0}_{m_0 i_0}$ and as $\bar{a}$ is an o-action, $\vdash_{R \cup R_0} Q \xrightarrow{\bar{a}} Q^{u_0}_{n_0 j_0}$. By **Com**$_c$ we deduce $\vdash_R P \mid Q \xrightarrow{\tau} P^{v_0}_{m_0 i_0} \mid Q^{u_0}_{n_0 j_0}$. Thus the original transition of *exp* has been matched by one of $P \mid Q$. The other transitions of *exp* can be shown to match in a similar fashion. ∎

THEOREM (Completeness). *For finite* CCS$^+$ *terms* P, Q, $P \sim Q$ *if and only if* $P = Q$.

Proof. The implication in one direction follows from Theorem 5.1. Conversely, suppose that $P \sim Q$. By Proposition 5.3, one can assume without loss of generality that P and Q are in strong normal form. Therefore, we show that

$$P \sim Q \Rightarrow P = Q \tag{5}$$

by induction on the maximum depth of P and Q, where P and Q and their subterms are in strong normal form.

Define

$$depth(\mathbf{0}) = 0, \qquad depth(\alpha.P) = 1 + depth(P),$$
$$depth\left(\sum_{i \in I} P_i\right) = max\{depth(P_i) : i \in I\}$$
$$depth\left(\sum_{i \in I} P_i\right) = max\{depth(P_i) : i \in I\}.$$

The base case of the induction, where the depth of P and Q is 0, is established easily, as then $P \equiv Q \equiv \mathbf{0}$ and $P = Q$ follows from law **A2**. Otherwise assume that P and Q have the form discussed below and, inductively, (5) holds for pairs of subterms of smaller depth. Suppose that

$$P \equiv \sum_{i \in I} P_i,$$

$$P_i \equiv \sum_{j \in J_i} P_i^j,$$

$$P_i^j \equiv \sum_{k \in K_{ij}} \alpha_{ik}^j . P_{ik}^j,$$

and

$$\vec{P}_i^{j_0} \equiv \sum_{j < j_0} P_i^j$$

for $i \in I$, $j_0 \in J_i$, and

$$Q \equiv \sum_{l \in L} Q_l,$$

$$Q_l \equiv \sum_{m \in M_l} Q_l^m,$$

$$Q_l^m \equiv \sum_{n \in N_{lm}} \beta_{ln}^m . Q_{ln}^m,$$

and

$$\vec{Q}_l^{m_0} \equiv \sum_{m < m_0} Q_l^m$$

for $l \in L$, $m_0 \in M_l$. We shall show that $Q + P_i = Q$ for all $i \in I$. It then follows that $Q + P = Q$ and by a symmetric argument that $P + Q = P$. Hence $P = Q$.

We prove inductively on j_0 that $Q + \vec{P}_i^{j_0} = Q$ for all $j_0 \in J_i$ and $i \in Y$, which, when we take j_0 to be the maximum element of J_i, yields $Q + P_i = Q$ for $P_i \in \mathscr{A}\mathscr{E}^+$. The base case when $j_0 = 0$ follows from **A4**, as then $\vec{P}_i^{j_0} \equiv \mathbf{0}$. Assume that $j_0 \in J_i$ and $j_0 > 0$ and inductively that $Q = Q + \vec{P}_i^{j_0}$. Because P is in strong normal form,

$$\vdash_R P \xrightarrow{\alpha} P_{ik}^{j_0},$$

where $\alpha_{ik}^{j_0} = \alpha \in (\mathscr{A} \cup \{\tau\})$, $\mathbf{ac}(\vec{P}_i^{j_0}) = A$ and $R = \bar{\mathscr{A}} - A$. As $P \sim Q$, there is a Q' such that

$$\vdash_R Q \xrightarrow{\alpha} Q' \quad \text{and} \quad P_{ik}^{j_0} \sim Q'.$$

It follows from the form of Q that $\beta_{ln_0}^{m_0} = \alpha \in (\mathscr{A} \cup \{\tau\})$ and $Q' \equiv Q_{ln_0}^{m_0}$ for some $l \in L$, $m_0 \in M_l$, and $n_0 \in N_{lm_0}$ such that

$$\mathbf{ac}(\vec{Q}_l^{m_0}) = A' \quad \text{and} \quad A' \cap R = \varnothing \quad (\text{i.e. } A' \subseteq A). \tag{6}$$

As $P_{ik}^{j_0}$, $Q_{ln_0}^{m_0}$ are bisimilar and have smaller depth than P, Q, we note that

$$P_{ik}^{j_0} = Q_{ln_0}^{m_0}. \tag{7}$$

By the splitting lemma, splitting the m_0th component of Q_l into $\beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}$ and $\sum_{n \neq n_0} \beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}$, we obtain

$$\begin{aligned} Q &= Q + (\vec{Q}_l^{m_0} \twoheadrightarrow \beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}) \\ &= Q + \vec{P}_i^{j_0} + (\vec{Q}_l^{m_0} \twoheadrightarrow \beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}) \\ &\qquad \text{by the inductive assumption,} \\ &= Q + (\vec{P}_i^{j_0} \twoheadrightarrow \beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}) + (\vec{Q}_l^{m_0} \twoheadrightarrow \beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}) \end{aligned} \tag{8}$$

The last step uses the topping lemma applicable as the acceptance set of $\vec{Q}_l^{m_0}$ is contained in the acceptance set of $\vec{P}_i^{j_0}$ (see 3). Hence

$$\begin{aligned} Q &= Q + (\vec{P}_i^{j_0} \twoheadrightarrow \beta_{ln_0}^{m_0} . Q_{ln_0}^{m_0}) \\ &\qquad \text{by (8)} \\ &= Q + (\vec{P}_i^{j_0} \twoheadrightarrow \alpha_{ik}^{j_0} . P_{ik}^{j_0}) \\ &\qquad \text{as } P_{ik}^{j_0} = Q_{ln_0}^{m_0}, \text{ noted in (7).} \end{aligned}$$

This has been shown for arbitrary $k \in K_{ij_0}$. Hence, by the use of **P4**, we obtain

$$\begin{aligned} Q &= Q + \left(\vec{P}_i^{j_0} \twoheadrightarrow \sum_{k \in K_{ij_0}} \alpha_{ik}^{j_0} . P_{ik}^{j_0}\right) \\ &= Q + \vec{P}_i^{j_0 + 1} \qquad \text{by definition.} \end{aligned}$$

Hence we have established the induction step. Therefore $Q = Q + P_i$ for all $i \in Y$. ∎

ACKNOWLEDGMENT

We are grateful to Claus Torp Jensen for helpful insights, and for his implementation of the Concurrency Workbench to handle agents with priority choice.

Received September 10, 1991; final manuscript received September 4, 1992

REFERENCES

1. Barrett, G. (1989), The semantics of priority and fairness in **occam**, *in* "Proceedings, MFPS 5, New Orleans."
2. Baeten, J. C. M., Bergstra, J. A., and Klop, J. W. (1985), "Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra," Technical Report CS-R8503, Center for Mathematics and Computer Science, Amsterdam.
3. Best, E., and Koutny, M. (1990), "Partial Order Semantics of Priority Systems," Technical Report 6/90, Institute of Computer Science, University of Hildesheim.
4. Brinksma, E. (1986), A tutorial on LOTOS, *in* "Protocol Specification, Testing, and Verification, V" (M. Diaz, Ed.), pp. 171–194, Elsevier, Amsterdam.

5. Camilleri, J. (1989), An operational semantics for **occam**, *Int. J. Parallel Programming* **18**, No. 5.
6. Camilleri, J. (1990), "Priority in Process Calculi," Ph.D thesis; Technical Report 227, Computer Laboratory, University of Cambridge.
7. Camilleri, J. (1991), A conditional operator for CCS, *in* "ConCur '91" (J. C. M. Baeten and J. F. Groote, Eds.), pp. 142–156, Lecture Notes in Computer Science, Vol. 527, Springer-Verlag, Berlin/New York.
8. Camilleri, J., and Winskel, G. (1991), CCS with priority choice, *in* "Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science."
9. Cleaveland, R., and Hennessy, M. (1990), Priorities in process algebras, *Inform. and Comput.* **87**, 58–77.
10. Costa, G., and Stirling, C. (1984), Weak and strong fairness in CCS, *in* "Mathematical Foundations of Computer Science" (M. P. Chytil and V. Koubek, Eds.), pp. 245–264, Lecture Notes in Computer Science, Vol. 176, Springer-Verlag, Berlin/New York.
11. Gerber, R., and Lee, I. (1990), CCSR: A calculus for communicating shared resources, *in* "ConCur '90" (J. C. M. Baeten and J. W. Klop, Eds.), pp. 263–277, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.
12. Groote, J. F. (1990), Transition system specifications with negative premises, *in* "ConCur '90" (J. C. M. Baeten and J. W. Klop, Eds.), pp. 332–341, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.
13. inmos (1984), "**occam** Programming Manual," International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ.
14. Jensen, C. T. (1991), The concurrency workbench with priorities, *in* "CAV '91" (K. Larsen and A. Skou, Eds.), pp. 147–157, Lecture Notes in Computer Science, Vol. 575, Springer-Verlag, Berlin/New York.
15. Jensen, C. T. (unpublished), Ph.D. thesis proposal, University of Aarhus, Denmark.
16. Milner, R. (1984), "Communication and Concurrency," International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ.
17. Smolka, S., and Steffen, B. (1990), Priority as extremal probability, *in* "ConCur '90" (J. C. M. Baeten and J. W. Klop, Eds.), pp. 456–466, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.
18. Tofts, C. (1990), A synchronous calculus of relative frequency, *in* "ConCur '90" (J. C. M. Baeten and J. W. Klop, Eds.), pp. 467–480, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.
19. Winskel, G., and Nielsen, M. (to appear), Models for concurrency, *in* "Hankbook of Logic in Computer Science" (S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, Eds.).