# A Semantics for Multiprocessor Systems

by

**Padmanabhan Krishnan**
Department of Computer Science
University of Canterbury
Christchurch 1, New Zealand
E-mail: paddy@cosc.canterbury.ac.nz

**Abstract**

In this paper we present a multiprocessor semantics for CCS [Mil80]. An operational semantics for processes under a finite number of processors is developed. The effect of adding or removing processors from the system is studied. A notion of strong bisimulation induced by the new semantics is defined. Issues related to a complete axiomatization of this congruence are examined and a complete equational system for a subset of CCS is presented.

## 1 Introduction

The idea of using observations or labeled transition systems as the basis for describing behaviors for concurrent systems is well known. However, most of the initial work for concurrent systems resulted in an 'interleaving semantics'. That is, parallelism was not distinguishable from non-determinism. Work by [DDM88] uses the notion of causality to present a non-interleaving semantics for CCS. [CH89] develop a theory based on the spatial distribution of processes. [KHCB91] uses the notion of location to develop a theory which accounts for the parallel nature of processes. While these theories differentiate parallelism and non-determinism, they do so only at the logical level. They do not study the behavior by 'actually executing' the process on a physical system. In other words, the architectural implications on behavior have not been addressed.

Given that there are many different types of architecture, it is only natural that a theory characterizing one system will not characterize another. The logical characterization can be thought of as the least common denominator; if processes identified by the theory will exhibit similar behavior on all systems which satisfy the assumptions of the theory. For example, if one considers only uniprocessor systems, parallelism will indeed be reduced to non-determinism. If one had an unbounded number of processors, behaviors consistent with pomset semantics [Pra86] could be exhibited. In real systems, it is not always possible to realize the architecture assumed by the theory. Resource limitations will induce restrictions on the possible behavior. Therefore it is necessary to index the behavior by the available resources.

In this paper we study the behavior of concurrent processes for a specific architecture, viz. shared memory systems. A shared memory system has a number of processors

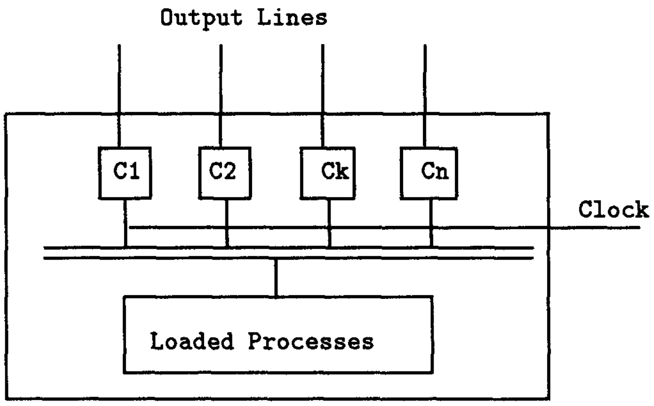

Figure 1: Machine Model

and a single memory unit which is accessed by the processor using a bus [JS80] (see figure 1).

This machine model is similar to the Chemical Abstract Machine [BB89]. The Chemical Abstract Machine models processes as being suspended in a solution with the ability to interact with one another. Our machine model can be considered to be a Chemical Abstract Machine with a bounded number of catalysts (processing elements) which are essential for any evolution.

The machine and language described in [BCM88] forms the basis for the semantics described here. We assume that the processors are homogeneous and memory is uniformly accessible to all processors. This allows 'logical migration', i.e., any process can use any processor. Scalability (the ability to add more processors) and fault-tolerance (the ability to function inspite of losing a processor) are important properties of a multiprocessor system. We consider the effect of adding/removing processors from a system on processes.

A theory for distributed memory systems has been studied in [Kri91] and is orthogonal to the work presented here. There the approach was to consider loosely coupled systems. The idea was to use a concept of location (introduced in [KHCB91]) to represent a virtual node. Processes were anchored to a particular location. Processes at different locations could evolve independently. Communication between locations was indicated by special asynchronous (i.e., had no complement in the CCS sense) message passing actions.

## 2 Multiprocessor Semantics

The language for which we develop a multiprocessor semantics is CCS [Mil80]. We first present an operational semantics based on labeled transitions systems for it. The semantics is indexed by a finite number of processors. Based on the operational semantics we develop a notion of bisimulation and relate the behaviors of processors with different numbers of processing elements. We also discuss the issues related to a complete axiomatization of the bisimulation equivalence.

As in CCS, we assume a set of actions $\Lambda$. As usual we assume $\bar{\cdot}$, to be a bijection on $\Lambda$ such that $\bar{\bar{a}} = a$. Typical elements of $\Lambda$ are denoted by $a, b \ldots$. A special action $\tau$ not in $\Lambda$ is used for synchronization.

The syntax of the language is as follows.

$$P = nil \mid a;P \mid P+P \mid (P \mid P) \mid (P \backslash a)$$

*nil* is a process which can exhibit no action, ';' denotes action or $\tau$ prefix, '+' non-determinism, '| ' parallel composition and '\' action restricting.

A structural operational semantics [Plo81] is defined as a generalization of the rules for CCS. We assume that the following black-box is a model of a multi-processor system which runs a process given n processing elements. There is a 'clock' line which when toggled advances each processing element by one step. The observer first toggles the 'clock' and then notes the behavior on the n-lines (which may appear at different times with respect to some real clock) and the process continues. This is shown in figure 1. The semantics developed here is similar to the step semantics developed in [vGV87] but the number of actions in a step is bounded. However as will be seen later, we do not assume a synchronous model. Therefore, our semantics is different from SCCS [Mil83].

Not all processors in the system may be required by a process at all the steps. For example, if a system has 2 processors to execute $a$;P only one of them can execute the action '$a$'. The other will necessarily be idle. (P may or may not be able to use both the processors.) Let $\delta$ represent idling (of a processor) and let Act $= \Lambda \cup \{\delta, \tau\}$

For the observations (the labels in the transition relation) we use n-tuples as opposed to multisets. This facilitates the requirement that synchronization of processes occur on the same processor. It would be unrealistic to assume synchronization across different processing elements. This captures the intuition that synchronization occurs at a location; the processor representing the location. Using the Chemical Abstract Machine analogy, synchronization can occur only by moving the processes physically close to each other. Architecturally, synchronization across processors would require the bus to support a particular protocol. It would be unrealistic to demand such a protocol for multiprocessor systems.

**Definition: 1** *Let $\mathcal{O}_n$ denote the function space from n to Act (or n-tuples) and for any $S \in \mathcal{O}_n$, Actions(S) = codomain(S).*

The intuition in using $\mathcal{O}_n$ is that if one is given $n$ processing elements one can observe n actions at every step.

| | |
|---|---|
| Prefix | $\forall\ 0 \leq \mathrm{i} \leq (\mathrm{n}\text{-}1)\ a;\mathrm{P} \xrightarrow{<\delta\ (i\ times)\ a\ \delta\ (n-1-i)\ times>}_n \mathrm{P}$ |
| Non-Determinism | $\dfrac{\mathrm{P} \xrightarrow{S}_n \mathrm{P}'}{\mathrm{P+Q} \xrightarrow{S}_n \mathrm{P}',\ \mathrm{Q+P} \xrightarrow{S}_n \mathrm{P}'}$ |
| Parallelism | $\dfrac{\mathrm{P} \xrightarrow{S1}_n \mathrm{P}',\ \mathrm{Q} \xrightarrow{S2}_n \mathrm{Q}',\ \ \mathrm{S} = \mathrm{S1} +_n \mathrm{S2}}{\mathrm{P} \mid \mathrm{Q} \xrightarrow{S}_n \mathrm{P}' \mid \mathrm{Q}'}$ |
| Interleaving | $\dfrac{\mathrm{P} \xrightarrow{S}_n \mathrm{P}'}{\mathrm{P} \mid \mathrm{Q} \xrightarrow{S}_n \mathrm{P}' \mid \mathrm{Q},\ \mathrm{Q} \mid \mathrm{P} \xrightarrow{S}_n \mathrm{Q} \mid \mathrm{P}'}$ |
| Hiding | $\dfrac{\mathrm{P} \xrightarrow{S}_n \mathrm{P}',\ a, \overline{a} \notin \mathrm{Actions(S)}}{\mathrm{P}\ \backslash a \xrightarrow{S}_n \mathrm{P}'\ \backslash a}$ |

Figure 2: Operational Semantics

Legal combinations of observations are defined as follows.

**Definition: 2** *Define a partial function $+_n$ on $\mathcal{O}_n \times \mathcal{O}_n \rightarrow \mathcal{O}_n$ as follows: $O1 +_n O2 = O$ where*

$$O(x) = \begin{cases} O1(x) & if\ O2(x) = \delta \\ O2(x) & if\ O1(x) = \delta \\ \tau & if\ O1(x) = \overline{O2(x)} \end{cases}$$

As processes can compete for the processors, one has to define consistency of processor allocation. We assume that only one action can be exhibited by a processor at any time. As mentioned earlier, if two processes are attempting to synchronize, they are required to be on the same processor.

An element of $\mathcal{O}_n$ can be thought of as observing n actions simultaneously. Thus $+_n$ defines combining observations in a truly parallel fashion. The definition requires a processor to be idle with respect to one process if the other is to be able to use it except in the case of synchronization. If both processes do not use a processor, it is idle in their combination. If both processes use the processor to exhibit unsynchronizable actions,, their parallel combination is undefined.

**Definition: 3** *Let $\longrightarrow_n \subseteq$ Processes $\times\ \mathcal{O}_n \times$ Processes, be the smallest relation satisfying the axioms in figure 2. It describes the behavior of processes when n processing elements are available.*

A brief and informal explanation of the operational semantics is as follows. The elementary action can be executed on any of the processors and due to sequentiality all but one will be idle. We do not require a process to be fixed to a processor. If the machine architecture is to be exploited, the migration of processes to different processors has to be permitted. An atomic action can be considered to be the basic unit of scheduling. The process is preempted after executing a single action and returned to the pool of processes competing for the limited resources.

Non-deterministic choice also has the usual definition; i.e., if a process can exhibit an action (or set of actions) so can its non-deterministic combination with other processes. The rules that determine the behavior under parallel composition are as follows. The first requires the assignment of processes P and Q to be compatible for the parallel composition to be successful. The second interleaves the execution. The rule for hiding is as usual; i.e., P $\backslash a$ cannot exhibit a behavior in which the action $a$ or $\overline{a}$ is involved.

It is possible to impose a step optimal parallelism requirement (under a limited number of processors) by requiring that all possible processor assignments fail before applying the interleaving law. This would be the adaptation of the maximal parallelism model [SM82] to suit limited resources. For example, one could require that the only acceptable behavior of (a|b) given 2 processors is executing them on different processors; interleaving is disallowed (i.e, "no unnecessary waiting" is modeled.) Interleaving would have to be permitted for a|b|c given 2 processors. However, this results in the parallel operator being not associative as shown in the example below.

**Example 1** *A possible behavior for the process (a | b) | c is the a and b followed by c. However a | (b | c) cannot exhibit this as (b | c) can only exhibit b and c.*

As this goes against the intuition of the parallel operator, the step optimal semantics is not adopted.

## 3 Strong Bisimulation

Park in [Par81] defines strong bisimulation, an equivalence relation on processes. That is, processes which have 'identical' operational behavior are equivalent. We define a generalization of strong bisimulation for defining equivalences between processes.

**Definition: 4** $P \precsim_n Q$ *iff* $P \xrightarrow{S1}_n P'$ *implies* $\exists\, Q'$: $Q \xrightarrow{S1}_n Q'$ *and* $P' \precsim_n Q'$.

In other words, P $\precsim_n$ Q if Q can exhibit all behaviors of P. We write $\sim_n$ for the equivalence induced by $\precsim_n$.

The properties of $\sim_n$ are similar to the CCS case.

**Proposition 1**

| | |
|---|---|
| $\sim_n$ *is a congruence* | $P+Q \sim_n Q+P$ |
| $(P+Q)+R \sim_n P+(Q+R)$ | $P+P \sim_n P$ |
| $P \mid Q \sim_n Q \mid P$ | $(P \mid Q) \mid R \sim_n P \mid (Q \mid R)$ |
| $P \mid nil \sim_n P$ | |

As only the parallel operator introduces multiple observations, it is natural that if a process P exhibits k non-idling actions, P must be composed of at least k parallel processes.

**Proposition 2** *If $P \xrightarrow{S}_n P'$ and the number of non-idling actions of S (i.e, cardinality of Actions(S)) is greater than 1, then there exists: 1) Processes P1, P2 and P3, 2) Observations S1 and S2 and 3) A subset of $\Lambda$ (possibly empty) H, such that:*
*1) $P1 \xrightarrow{S1}_n P1'$, 2) $P2 \xrightarrow{S2}_n P2'$, 3) S1+S2=S and*
*4) Either $(P1' \mid P2' \mid P3) \sim_n P'$ (H is the empty set) or $((P1' \mid P2' \mid P3) \backslash H) \sim_n P'$*

**Proof Outline:** By induction on the structure of the process. Let P be (R1 | R2). In this case H will be the empty set. If both R1 and R2 contribute to form S then P1 is R1, P2 is R2, and P3 is *nil* . If only one evolved say R1, then by the induction hypothesis, there are $R_{11}$, $R_{12}$ and $R_{13}$, such that $R_{11} \xrightarrow{S1}_n R'_{11}$ and $R_{12} \xrightarrow{S2}_n R'_{12}$ and $R'_{11} \mid R'_{12} \mid R_{13} \sim_n R1'$. Now $P' \sim_n (R1' \mid R2)$. Then letting P1 be $R_{11}$, P2 being $R_{12}$ and P3 being $(R_{13} \mid R2)$ satisfies the condition.

If P is of the form $(R1 \mid R2)\backslash H_1$, the above argument is valid but with H equal to $H_1$.
□

Note that in the above result we do not derive the structure for P, as P could have made various choices and one has to introduce choices at every point where an action prefix occurs. For example,

( (((a;P1 + P1′) | (c;Q1 + Q1′)) + R1′) | (((b;P2 + P2′) | (d;Q2 + Q2′)) + R2′) | P3) + P4′

under 4 processors and the observation $< a, b, c, d >$ requires the introduction of P1′, Q1′ etc. (which may be nil). While this can be done in principle it is not very illuminating.

It is also easy to see that if a set of actions is exhibited by a process, any non-idling subset of it can also be exhibited.

**Definition: 5** *Let R and S $\in \mathcal{O}_n$. Define $R \leq S$, iff there is a 1-1 map F, on {1 .. n} such that $\forall$ i, $R(i) \neq \delta$ implies $R(i) = S(F(i))$. i.e., S observes more actions but with possibly different processor usage.*

**Proposition 3** *If $P \xrightarrow{S}_n P'$ and $R \leq S$, and $\exists$ i, $R(i) \neq \delta$ , then $\exists$ P″ such that P $\xrightarrow{R}_n P''$*

**Proof** By structural induction. □

CCS has an expansion theorem (i.e., reduction of parallelism to non-determinism). For example, (a | b) $\sim_{CCS}$ (a;b+b;a), and one would expect a similar law for the n-processor case. The expansion theorem could be expected to be a reduction of a process which can exhibit $n+1$ actions, but is given only $n$ processors, to a process which can exhibit only n actions. But unfortunately that is not the case.

**Example 2** *Consider P= (a | b | c) given 2 processors. If it is bisimilar to a term T then T can exhibit all the 3 actions in one step given 3 processes. The argument is as follows. Assume T cannot exhibit the 3 actions in one step. As P can exhibit a and evolve to the process (b | c), T could involve terms such as (a;b) | c or a;(b | c). The first type is disallowed as it can exhibit c and evolve to (a;b). But no c evolution of P is bisimilar to (a;b). The second type term is not sufficient as P can exhibit action a and b in one step.*

The lack of an expansion theorem for P can formally be stated as follows.

**Proposition 4** *Let P = (a | b | c). If P $\sim_2$ Q+R, then either P $\sim_2$ Q or P $\sim_2$ R.*

The intuition behind this result is that the | combinator does not force both its branches to evolve. As the transition rule for parallel composition permits interleaving, it is impossible to force a process to exhibit multiple actions at a particular step. This problem also prevents the axiomatization of the $n$ processor bisimulation. In section 4 we describe how this drawback can be overcome.

Our semantics is a generalization of the standard CCS semantics by explicitly considering the number of processors in the system. Clearly, if there is only one processor in the system, the standard behavior must be exhibited. This is indeed the case.

**Proposition 5** $\sim_1 \; = \sim_{CCS}$ .

**Proof:** It is easy to verify that $\longrightarrow_1$ is identical to the $\longrightarrow$ rules for CCS. □

As we have $n$ processing elements, we develop a theory relating processes and processors. It is easy to see that if two processes are similar under n+1 processors, they will be related under n processors.

**Proposition 6** *$P \overset{\sqsubset}{\sim}_{n+1} Q$ implies $P \overset{\sqsubset}{\sim}_n Q$.*

**Proof:** From proposition 3.

Clearly P $\overset{\sqsubset}{\sim}_n$ Q then P $\overset{\sqsubset}{\sim}_{n+1}$ Q, does not hold as by adding more resources one can expose 'true concurrency'. For example, (a | b) $\overset{\sqsubset}{\sim}_1$ (a;b + b;a), but (a | b) $\not\overset{\sqsubset}{\sim}_2$ (a;b + b;a). However, if the process on the right is the 'more parallel one', the result holds.

**Proposition 7** *If Q is a process not involving +, $P \stackrel{\sqsubset}{\sim}_n Q$ implies $P \stackrel{\sqsubset}{\sim}_{n+1} Q$.*

**Proof Outline:** Let Q have no +, P $\stackrel{\sqsubset}{\sim}_n$ Q but P $\not\stackrel{\sqsubset}{\sim}_{n+1}$ Q. As P $\not\stackrel{\sqsubset}{\sim}_{n+1}$ Q, either there is a transition P $\xrightarrow{S}_n$ P′ and Q has no transition labeled by S or Q $\xrightarrow{S}_n$ Q′ and P′ $\not\stackrel{\sqsubset}{\sim}_{n+1}$ Q′. Consider the first case. It is clear that the cardinality of S is n+1 (if less than n it violates P $\stackrel{\sqsubset}{\sim}_n$ Q). Thus, by proposition 2 S is composed of S1 and S2 such that P $\xrightarrow{S1}_n$ and P $\xrightarrow{S2}_n$. As the cardinality of S1 and S2 is less than n+1, Q $\xrightarrow{S1}_n$ and Q $\xrightarrow{S2}_n$. If Q cannot exhibit S, then either 1) S1+S2 is not defined which is not the case or 2) there is a choice between S1 and S2 in which case Q has a +. □

## 4 Axiomatization

In this section we discuss the issues related to the axiomatization of finite processes of the bisimulation equivalence for *n* processors. For the moment consider the language without hiding. Consider the set of equations in figure 3.

| | |
|---|---|
| P + P = P | P + *nil* = P |
| P + Q = Q + P | P \| Q = Q \| P |
| (P + Q) + R = P + (Q + R) | (P \| Q) \| R = P \| (Q \| R) |
| (P \| *nil* ) = P | |

Figure 3: Tentative Equations

The parallel axioms are necessary as (a | b) $\sim_2$ (b | a), but cannot be decomposed into various components. However, this set of axioms is not complete. For example, (a | b) $\sim_2$ (a | b) + a;b cannot be proved. Furthermore, the lack of an expansion theorem (as explained via an example) is not satisfactory. That is, (a | b | c) under two processors will exhibit some interleaving and is in 'normal form'.

The principal problem is that | is too 'powerful'. It permits any non-empty subset of the actions that can be exhibited in one step. Therefore, it is essential to have a construct which forces multiple actions to be performed in one step. For this we alter a single action prefix to a multiset prefix. A multiset captures multiple actions that occur in one step. Interleaving of the actions within a multiset is *not* permitted. That is, if the cardinality of the multiset is greater than the number of available processors no evolution is possible.

This can be used to model parallelism. For example, (a | b) can be considered to be an abbreviation for a;b + b;a + {a,b}. If there is only one processor {a,b} cannot contribute to the behavior and ( a | b) is equivalent to a;b+b;a. Similarly, (a | b | c) can be thought of as a;(b | c)+ b;(a | c)+ c;(a | b)+ {a,b};c + {a,c};b + {b,c};a + {a,b,c} and if there are only 2 processors, {a,b,c} will not contribute to the behavior.

Thus a multiset prefix represents 'forced' parallelism. Therefore, for a complete axiomatization of the bisimulation equivalence the appropriate generalization of CCS for the multiprocessor case is: 1) Observing multiple actions and 2) Replacing the single action prefix by a multiset prefix.

In the rest of this section we show that if the language permits a multiset prefix, the resulting bisimulation equivalence for finite processes can be completely axiomatized. We also assume that the number of processors is fixed ($n \geq 1$).

**Definition: 6** *Define a multiset m as a function, m: Act* $\rightarrow \mathcal{N}$

*Define the cardinality of a multiset m,* $\mid m \mid$ *, as* $\sum_{a \in Act} m(a)$ *where* $\sum$ *indicates integer addition.*

The following is the syntax for a multiprocessor language whose bisimulation semantics is axiomatized.

$$\mathrm{P} = nil \mid \mathrm{ms;P} \mid (\mathrm{P} \mid \mathrm{P}) \mid (\mathrm{P} + \mathrm{P}) \mid (\mathrm{P} \setminus a)$$

The only difference from the initial language is that action prefix ($a$) is replaced by a multiset prefix (ms). The semantics of an atomic action permitted the use of any of the available processors. Similarly the semantics of a multiset of actions permits any possible assignment of processors to the actions. The multiset prefix introduces another level of scheduling. Given a multiset an allocation of actions to processors is required. This is defined by the function *Assign*, which behaves as follows. Given an empty set, all the processors in the system are idle and that is the only possible assignment. Given an assignment of k actions, the k+1st action can be scheduled on any of the idle processors. Complementary actions within a multiset prefix cannot synchronize with one another. For example, if m is a multiset such that m($a$)=1 and m($\overline{a}$)=1, *Assign* will require at least two processors to execute it.

**Definition: 7** *Assume a fixed n. Assign is the smallest set satisfying the following*

- *Assign* $\emptyset = \{ < \delta , \dots ,\delta > \}$

- *If* $(Y \in \mathit{Assign}\ m)$ *and* $Y(i) = \delta$ *and*

$$X(j) = \begin{cases} a & \text{if } j = i \\ Y(j) & \text{otherwise} \end{cases} \text{ and}$$

$$m'(\mu) = \begin{cases} m(\mu) + 1 & \text{if } \mu = a \\ m(\mu) & \text{otherwise} \end{cases}$$

*then* $X \in$ *Assign* $m'$.

Given an observation, the multiset that gave rise to it can be obtained by the function $\mathit{Assign}^{-1}$ defined as follows.

**Definition: 8** *$Assign^{-1}(S) = m$ such that $m(a) = cardinality(\{i$ such that $S(i) = a\})$*

As *Assign* permits all possible allocations of actions to processors, the following hold.

**Proposition 8** *If $S \in Assign(m)$ and $S'$ is a permutation of $S$ then $S' \in Assign(m)$.*

**Proposition 9** *If $S \in Assign(m)$ then $Assign^{-1}(S) = m$.*

**Example 3** *Consider a 2 processor system. If $m(a)=1, m(b)=1$ then Assign $m = \{ <a,b>, <b,a> \}$. $Assign^{-1}(<a,b>) = \{a,b\}$.*

The semantics of multiset prefix (ms;P) is given in figure 4. The transition rules for the other constructs are as before.

| | |
|---|---|
| Multi-set Prefix | $\dfrac{S \in \text{Assign(ms)}}{\text{ms;P} \xrightarrow{S}_n \text{P}}$ |

Figure 4: Operational Semantics for Multiset Prefix

Define strong bisimulation equivalence for the language as before. The principal aim of considering a language with multi-set prefixes is to be able to have an axiomatization of bisimulation. To do this we need a generalization of the expansion theorem. The CCS version needs to be generalized not only to handle multiset prefixes but also to combine multiset prefixes from two processes to form another prefix.

Towards that aim we define the functions *Combine* and *Choice*. *Combine* m1 m2 as the set of all possible behaviors that can result by exhibiting the multisets m1 and m2 in one step. *Choice* is used by *Combine* to synchronize two elements to exhibit $\tau$.

**Definition: 9** *Combine of two multisets is the smallest set satisfying the following conditions.*

- *Combine $\emptyset$ m1 = Combine m1 $\emptyset$ = \{ m1 \}*

- *If $m1(a) \neq 0$ and $m2(\bar{a}) = 0$ then*
  *Combine m1 m2 = \{ $S \cup \{ <a, m1(a)> \}$ for $S \in$ Combine m1' m2 where*
  *$m1' = m1(\lceil\ (dom(m1)\text{-}\{a\}) \}$*

- *If $m1(a) = k1$ and $m2(\bar{a}) = k2$ then*
  *Combine m1 m2 = \{ $S \cup D$ where $S \in$ Combine m1' m2',*
  *$m1' = m1(\lceil\ (dom(m1)\text{-}\{a\}$, $m2' = m2\ (\lceil\ (dom(m2)\text{-}\{\bar{a}\})$ and*
  *$D \in$ Choice m1(a) m2(a) a \}*

| | |
|---|---|
| $P + P = P$ | $P + nil = P$ |
| $(P + Q) + R = P + (Q + R)$ | $(m;P) \backslash a = nil$ if $m(a)$ or $m(\bar{a}) \neq 0$ |
| $(m;P) \backslash a = m;(P\backslash a)$ if $m(a)$ and $m(\bar{a}) = 0$ | $(P + Q) \backslash a = (P \backslash a) + (Q \backslash a)$ |
| $(m;P) = nil$ if $\mid m \mid > n$. | $nil \backslash a = nil$ |

Figure 5: Equations

- *Choice k1 k2 a* $= \{ \{ <\tau, i>, <a, k1-i>, <\bar{a}, k2-i> \}$ *where* $0 \leq i \leq min(k1,k2)$ $\}$

Two multisets can be combined to yield all possible synchronizations (including none). For example, $\{a,b\}$ $\{\bar{a},\bar{b}\}$ can result in $\{ a,b,\bar{a},\bar{b} \}$ or $\{ a, \tau,\bar{a} \}$ or $\{ b, \tau,\bar{b} \}$ or $\{ \tau, \tau\}$. The first being no synchronization, the second the synchronization of b, the third the synchronization of a and the fourth, both a and b are synchronized. Not all combinations may contribute to legal behavior. In the above example if there are only 2 processors, only the last combination can be observed. Note that in the CCS case, actions can only be combined to yield a set of cardinality 1, viz., only $\tau$ is legal.

We should remark that the multiset prefix could have been replaced by a tuple-prefix without affecting the completeness results. For example, $\{ab\}$;P (which is multiset prefix) can be represented as $(\langle ab\rangle$;P + $\langle ba\rangle$;P) in the tuple-prefix. The tuple-prefix representation does not require the auxiliary definitions *Assign*, *Combine* and *Choice*. However, the representation is more concrete than the multiset form. Given the usefulness of multisets for multiprocessor systems [BCM88], we use the multiset prefix.

## 4.1 Completeness

Having defined the auxiliary functions, we can now present a set of axioms which completely axiomatize bisimulation equivalence for multiset prefix CCS. As the operational semantics was defined for a fixed n, the set of axioms also assumes a fixed n. The proof technique for CCS is adequate. That is, we define a normal form, show that all finite process can be reduced to normal form and via an absorption lemma we show that the set of axioms is complete.

Consider the equations defined in figure 5 (the usual axioms) and 6 (the expansion theorem).

**Proposition 10** *The set of axioms is sound; that is* $P = Q$ *implies that* $P \sim_n Q$.

**Proof** Standard.

The proof of completeness involves the definition of a normal form, then showing that all process can be proved to have a normal form and if two processes are bisimilar, they

$$\boxed{\begin{array}{c}\text{If } P = \sum_i m_i;P_i \text{ and } Q = \sum_j m_j;Q_j \text{ and } C_{i,j} = \text{Combine } m_i\ m_j \text{ then} \\ (P \mid Q) = \sum_i m_i;(P_i \mid Q) + \sum_j m_j;(P \mid Q_j) + \sum_i \sum_j \sum_{m \in C_{i,j}} m;(P_i \mid Q_j)\end{array}}$$

Figure 6: Expansion Theorem

can be proved to have identical normal forms. The proofs are only outlined as the proof techniques are well known.

**Definition: 10** *Define a process to be in normal form if it is of the form* $\sum_i m_i; P_i$ *and each* $P_i$ *is in normal form. and for all i,* $\mid m_i \mid \ \leq n$

**Proposition 11** *All process can be reduced to normal form using the equational rules.*

**Proof:** By induction on the size of the process. □

**Proposition 12** (**Absorption Lemma**) *Let P be in normal form. If* $P \xrightarrow{S}_n P'$ *and* $P' = Q$ *then P + m1;Q = P where m1 =* $Assign^{-1}(S)$

**Proof:** Let P = $\sum_i m_i; P_i$ If P $\xrightarrow{S}_n$ P′ then $\exists i, S \in Assign(m_i)$ and P′ identical to $P_i$. Hence, P + $m_i$;Q = P. □

**Proposition 13** *The set of axioms is complete; i.e.,* $P \sim_n Q$ *implies* $P = Q$.

**Proof:** It is sufficient to consider only normal forms as all processes can be reduced to normal form. We prove by induction on the length of the normal forms. Let P = $\sum_{i \in I} m_i; P_i$ and Q = $\sum_{j \in J} m'_j; Q_j$ such that P $\sim_n$ Q. We show that this implies P = P + Q = Q. To prove P + Q = P, it is sufficient to show $\forall j$, P + $m'_j; Q_j$ = P. As P $\sim_n$ Q, there is a $m_i$ equal to $m'_j$ and $P_i \sim_n Q_j$. Furthermore, $P_i = Q_j$. Therefore, from the absorption lemma P + $m'_j; Q_j$ = P. □

## 5 Conclusion

We have presented a semantics for multiprocessor CCS. The axiomatization of the bisimulation equivalence required the introduction of multi-set prefixes. The analogy between the expansion theorem for CCS and multiprocessor CCS is that in CCS | was translated to choice with action prefix, while in multiset CCS | was translated to choice with multiset prefix. From a programming view point, the user can use CCS, a compiler for a multiprocessor system will convert it to CCS with multiset prefix and a scheduler (for a particular machine) will ignore certain multisets (due to cardinality) and make the processor assignments.

As mentioned in the introduction, there are a number of non-interleaving semantics for concurrency [DDM88, BB89, BC87]. Current work is on in trying to prove a "limiting" theorem, i.e., given sufficient number of processors, the semantics in this paper coincides with the other semantics.

## Acknowledgment

The author thanks Uffe Engberg, Jens Palsberg, Peter Mosses and Rod Harries for their comments and encouragement. Many thanks to the anonymous referees for several useful suggestions.

## References

[BB89] G. Berry and G. Boudol. The Chemical Abstract Machine. Technical Report 1133, INRIA-Sophia Antipolis, December 1989.

[BC87] G. Boudol and I. Castellani. On Semantics of Concurrency: Partial Orders and Transition Systems. In *Proceedings of the International Joint Conference on TAPSOFT: LNCS 249*. Springer Verlag, 1987.

[BCM88] J. P. Banatre, A. Coutant, and D. Metayer. A Parallel Machine for Multiset Transformation and its Programming Style. *Future Generation Computer Systems*, 4:133–144, 1988.

[CH89] I. Castellani and M. Hennessy. Distributed Bisimulations. *Journal of the Association for Computing Machinery*, 36(4):887–911, October 1989.

[DDM88] P. Degano, R. DeNicola, and U. Montanari. A Distributed Operational Semantics for CCS Based on Condition/Event Systems. *Acta Informatica*, 26:59–91, 1988.

[JS80] A. Jones and P. Schwarz. Experience using multiprocessor systems — A status report. *ACM Computing Surveys*, 12(2), 1980.

[KHCB91] A. Kiehn, M. Hennessy, I. Castellani, and G. Boudol. Observing localities. In *Mathematical Foundations of Computer Science(MFCS)*, 1991.

[Kri91] P. Krishnan. Distributed CCS. In *Theories of Concurrency: Unification and Extension: CONCUR-91, LNCS:527*, August 1991.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Lecture Notes on Computer Science Vol. 92. Springer Verlag, 1980.

[Mil83] R. Milner. Calculus for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In *Proceedings of the 5th GI Conference, LNCS-104*. Springer Verlag, 1981.

[Plo81] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pra86] V. Pratt. Modelling Concurrency with Partial Orders. *International Journal of Parallel Programming*, 15(1), 1986.

[SM82] A. Salwicki and T. Muldner. On the Algorithmic Properties of Concurrent Programs. In *LNCS-125*. Springer Verlag, 1982.

[vGV87] R. J. van Glabbeek and F. W. Vaandrager. Petri Net Models for Algebraic Theories of Concurrency. In J. W. deBakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE-II , LNCS 259*. Springer Verlag, 1987.