



CCS Expressions, Finite State Processes, and Three Problems of Equivalence

Paris C. Kanellakis Scott A. Smolka
Department of Computer Science
Brown University
Providence, Rhode Island 02912

Abstract

We examine the computational complexity of testing finite state processes for equivalence, in the Calculus of Communicating Systems (CCS). This equivalence problem in CCS is presented as a refinement of the familiar problem of testing whether two nondeterministic finite state automata (n.f.s.a.) accept the same language. Three notions of equivalence, proposed for CCS, are investigated: (1) observation equivalence, (2) congruence, and (3) failure equivalence. We show that observation equivalence (\approx) can be tested in cubic time and is the limit of a sequence of equivalence notions (\approx_k), where, \approx_1 is the familiar n.f.s.a. equivalence and, for each fixed k , \approx_k is PSPACE-complete. We provide an $O(n \log n)$ test for congruence for n state processes of bounded fanout, by extending the algorithm that minimizes the states of d.f.s.a.'s. Finally, we show that, even for a very restricted type of process, testing for failure equivalence is PSPACE-complete.

1. Introduction

The *Calculus of Communicating Systems* (CCS) is an elegant formalism [M1,M2,M3], for the description and manipulation of concurrent systems. Together with other new formalisms, such as, the *failures* model for *Communicating Sequential Processes* (CSP) in [HBR,B], it represents the state of the art on modeling concurrent computation. One of the nicest

features of CCS is that it is an algebraic theory, that closely parallels the classical algebraic theory of regular expressions and finite state automata [M2].

CCS is based on two central ideas [M1]:

- (1) The notion of *observation equivalent processes* (i.e., processes that are indistinguishable to an observer). Equivalence classes of processes are the basic objects of CCS.
- (2) The *definition* and *manipulation* of these basic objects using *algebraic operators* (e.g., U , \cdot , $*$, $|$). Of these operators, *composition* $|$ captures many features of interleaving of concurrent computations, and of message passing.

In this paper we focus on the notion of *observation equivalence*. We examine it from the point of view of computational complexity and in the context of *finite state processes*. We also investigate other notions, such as, *congruence* [M2] and *failure equivalence* [HBR], which have been proposed as alternatives to observation equivalence, perhaps more natural for certain types of processes.

In our exposition we stress the similarity of CCS to the classical theory of regular expressions, in order to show how new problems, which are meaningful in the context of distributed computing, can be derived from classical problems. As in [M2], we only deal with a very restricted set of algebraic operators of the calculus (U , \cdot , $*$), that produce the *star expressions* in CCS. These expressions are syntactically identical with regular expressions, but instead of having semantics "sets of strings", their meaning is "equivalence classes of processes" (see Section 2.3 and Figure 3).

We first rigorously define all relevant features from CCS, in a fashion similar to [M2]. We consider (Section 2.1) finite state processes, slightly more general than the familiar nondeterministic finite state automata with empty moves (n.f.s.a.'s) [HU] and some useful subsets of these; e.g., *standard* processes or n.f.s.a.'s, *observable* processes without unobservable or

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1983 ACM 0-89791-110-5/83/008/0228 \$00.75

τ -moves, *restricted* processes with all states accepting states. We review three basic notions of equivalence, i.e., observation equivalence, congruence, and failure equivalence, and the relationships between them. We treat these notions as refinements of the classical notion of equivalence on the basis of accepting the same language (Section 2.2). Finally (Section 2.3) we relate star expressions in CCS to regular expressions and pose the CCS-EQUIVALENCE problem for star expressions. This is, in essence, a problem of testing processes for observation equivalence, or congruence, or failure equivalence, depending upon the chosen notion of equivalence.

Congruence, which equals observation equivalence for observable finite state processes, can be decided efficiently. Moreover, by generalizing the algorithm of [H], which finds the minimum state deterministic finite automaton, we have an $O(c^2 n \log n)$ test of congruence, for n state observable processes of fanout c . This also resolves a combinatorial partitioning problem of independent interest.

An important property of observation equivalence is that, unlike classical n.f.s.a. equivalence [StM], it is efficiently decidable in cubic time. Moreover, we show that it is the limit of a sequence of refinements of equivalence notions \approx_k of which \approx_1 is n.f.s.a. equivalence. We show that \approx_k is PSPACE-complete for every fixed k , a complexity that disappears when we take limits. Our negative results hold even for the restricted and observable model.

For failure equivalence, an equivalence notion between \approx_1 and \approx_2 we show PSPACE-completeness, even for the restricted and observable model.

Section 2 contains the model, Section 3 the algorithms for congruence and the partitioning problem, Section 4 the analysis of observation equivalence, and Section 5 the analysis of failure equivalence.

We would like to point out that we examine only one critical feature of CCS, namely, the choice of equivalence notion. We do not discuss other important algebraic features of CCS, such as, (a) the composition operator $|$, or (b) the role of τ -moves (which we certainly define as unobservable moves of processes, but which become practically significant only in the presence of $|$). However, we believe that we clarify some useful complexity questions in the calculus. In addition, in Section 6, we highlight the CCS-EQUIVALENCE problem for extended star expressions. This is the problem examined in this paper, extended with the composition operator but without τ -moves; an interesting and possibly efficiently solvable combinatorial problem.

2. The Model

2.1 Finite State Processes

The basic building block of our model for distributed computation is the finite state process, which very much resembles the nondeterministic finite state automaton (n.f.s.a.) of the classical theory of computation.

Definition 1: For a set of symbols Σ (called *actions*), and a special symbol τ not in Σ (called the *unobservable action*), and a set of symbols V (called *variables*), we define a *Finite State Process* (F.S.P.) as a quadruple $\langle K, p_0, \Delta, E \rangle$, where:

- 1) K is a finite set of *states*.
- 2) $p_0 \in K$ is the *start state*.
- 3) $\Delta: K \times (\Sigma \cup \{\tau\}) \rightarrow 2^K$ is a function called the *transition function*.
- 4) $E: K \rightarrow 2^V$ is a function called the *extension function*. \square

A F.S.P. can be represented, in the obvious fashion, as a directed graph with labels on the arcs (i.e. transitions) from $\Sigma \cup \{\tau\}$ and labels on the nodes from 2^V . The various extensions (i.e. the labels on the nodes) represent different flavors of acceptance and are used in [M2]. These extensions are the only difference from the classical notion of a n.f.s.a. with empty transitions [HU].

We call this the *general* model of F.S.P.'s. If no τ -transitions are present we have the *observable model* [M2], and if, in this case, there is exactly one transition for each symbol in Σ we have the *deterministic* model. On the other hand, for $|V| = 1$, we have the classical n.f.s.a., with empty transitions, which we call the *standard* model. If in the standard model, for all states p , $E(p) = V = \{X\}$ (i.e. all states are accepting states, but there could be some missing transitions), we have the *restricted* model. Nontrivial subsets of the restricted model are the *restricted observable* model (called *r.o.u.* when $|\Sigma| = 1$), and the model where the F.S.P. is a *finite tree* rooted at p_0 . The hierarchy of models is depicted in Figure 1a. Every one of the these models corresponds to some nontrivial case in our exposition. Examples of such F.S.P.'s are presented in Figure 1.b (we use standard notation from [HU] for the standard model).

We will always deal with *states of F.S.P.'s* (e.g. equivalent states) and the processes to which these states belong will be clearly defined from the context.

Let $s \in \Sigma^*$ and p, p' be states of a F.S.P.. If $s = \epsilon$, we say that $p \xRightarrow{s} p'$, when there is a sequence of k arcs in the graph of the F.S.P. from p to p' with labels τ^k , $k \geq 0$. (Obviously always $p \xRightarrow{s} p$.) If $s = s_1 s_2 \dots s_n$, $s_i \in \Sigma$, $1 \leq i \leq n$, we say that $p \xRightarrow{s} p'$, when there is a sequence of $k_0 + k_1 + \dots + k_n + n$ arcs in the graph of the F.S.P. from p to p' with labels $\tau^{k_0} s_1 \tau^{k_1} s_2 \dots s_n \tau^{k_n}$, $k_0, k_1, \dots, k_n \geq 0$. We use a different symbol for τ and the empty string ϵ , because of the particular role of τ as the unobservable action in distributed computation [M1].

In the *restricted* model of F.S.P.'s, the only feature that distinguishes states is the absence

of certain transitions. For this important case, let p be a state and $s \in \Sigma^*$. We say that $p \xRightarrow{s} \text{dead}$, when there is no p' such that $p \xRightarrow{s} p'$. This concept is formalized in [HBR] as the *failures*(p) for state p in the restricted case:

$\text{failures}(p) = \{(s, Z) \mid s \in \Sigma^*, Z \subseteq \Sigma \text{ with the property: there is a state } q, \text{ such that for every } z \text{ in } Z: p \xRightarrow{s} q \text{ and } q \xRightarrow{z} \text{dead}\}$

For example, assuming $\Sigma = \{a, b, c\}$, the failures for the start state of the finite tree process of Figure 1.b are:

$$\{\epsilon\} \times 2^{\{b, c\}} \cup \{a\} \times 2^{\{a\}} \cup \{ab\} \times 2^\Sigma \cup \{ac\} \times 2^\Sigma$$

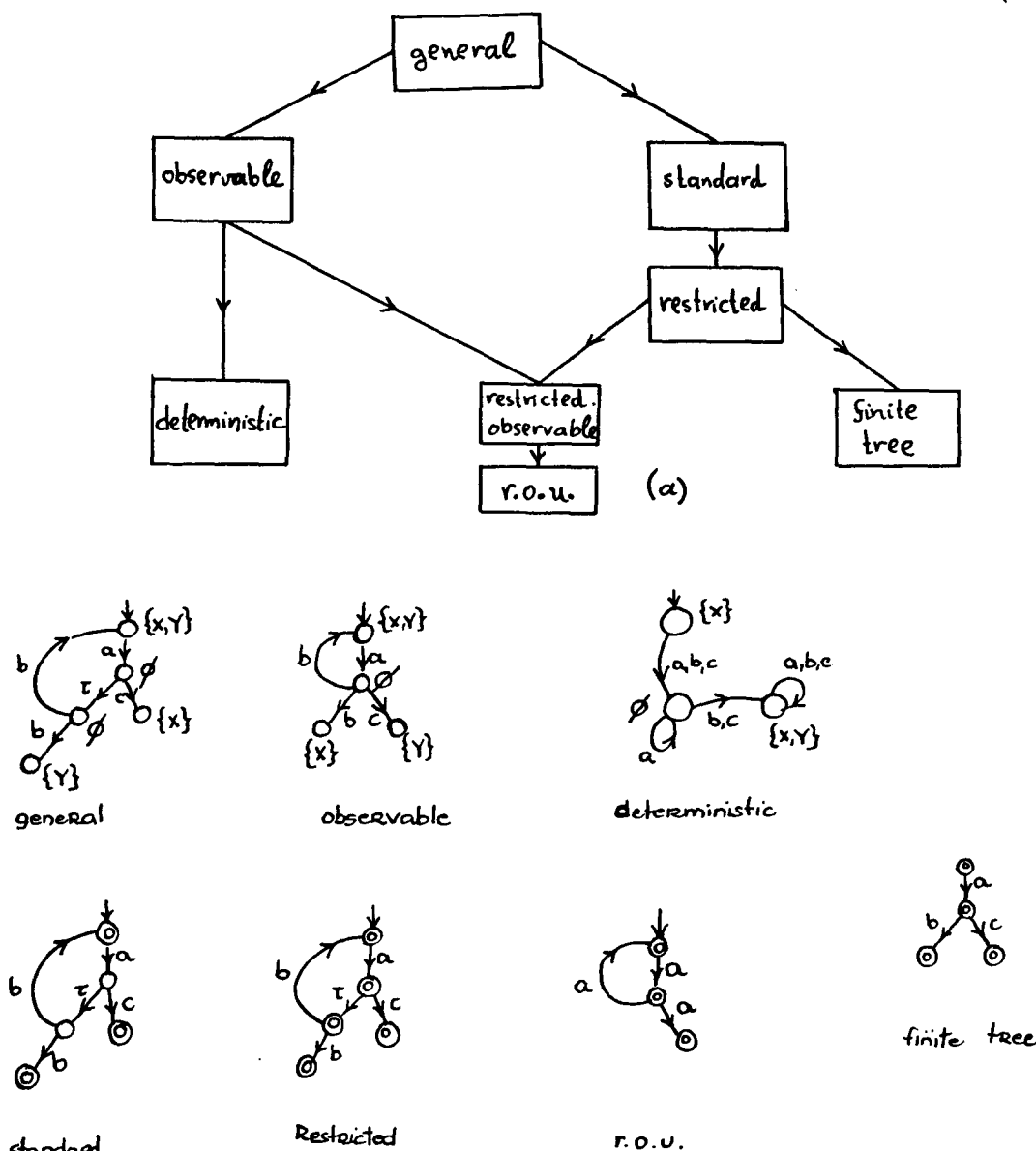


Figure 1

(b)

2.2 Equivalence of F.S.P. States

The essential new idea in CCS is a new notion of equivalence between states of F.S.P.'s. A number of "candidates" for the correct notion of equivalence have been proposed and investigated [M1,M2,M3,HBR,B]. We will deal with three such notions: *observation equivalence*, *congruence*, and *failure equivalence*. The expositions of [M1,HBR] are, to a great extent, devoted to establishing the practical relevance of such choices in the context of distributed computation.

In general, for two F.S.P. states p, q to be "equivalent", it is not enough to say that they represent start states of n.f.s.a.'s accepting the same language (as in the classical case). In particular:

Definition 2: Two F.S.P. states p, q in the general model are *k-observation equivalent* ($p \approx_k q$) iff:

1. $E(p) = E(q)$ when $k = 0$
2. For every $s \in \Sigma^*$ when $k > 0$
 if $p \xrightarrow{s} p'$ then $(\exists q': q \xrightarrow{s} q' \text{ and } p' \approx_{k-1} q')$
 if $q \xrightarrow{s} q''$ then $(\exists p'': p \xrightarrow{s} p'' \text{ and } q'' \approx_{k-1} p'')$

States p, q are *observation equivalent* [M1] ($p \approx q$) iff $p \approx_k q$ for all $k \geq 0$. \square

Definition 3: Two F.S.P. states p, q in the general model are *k-limited observation equivalent* ($p \approx_k q$) iff:

1. $E(p) = E(q)$ when $k = 0$
2. For every $s \in \Sigma \cup \{\varepsilon\}$ when $k > 0$
 if $p \xrightarrow{s} p'$ then $(\exists q': q \xrightarrow{s} q' \text{ and } p' \approx_{k-1} q')$
 if $q \xrightarrow{s} q''$ then $(\exists p'': p \xrightarrow{s} p'' \text{ and } q'' \approx_{k-1} p'')$

States p, q are *limited-observation equivalent* ($p \approx q$) iff $p \approx_k q$ for all $k \geq 0$.

In the observable model, we denote $p \approx q$ as $p \sim q$ and call p, q *congruent* [M2]. \square

Definition 4: Two F.S.P. states p, q in the restricted model are *failure equivalent* ($p \equiv q$) [HBR] iff:

$$\text{failures}(p) = \text{failures}(q). \quad \square$$

Let $\Lambda \subset \Sigma^*$ be a set of strings. We call a relation R between F.S.P. states a Λ -*fixpoint* when

$p R q$ iff

1. $E(p) = E(q)$
2. For every $s \in \Lambda$
 if $p \xrightarrow{s} p'$ then $(\exists q': q \xrightarrow{s} q' \text{ and } p' R q')$
 if $q \xrightarrow{s} q''$ then $(\exists p'': p \xrightarrow{s} p'' \text{ and } p'' R q'')$

Lemma 1: In the general model for F.S.P.'s:

- (a) \approx is a Σ^* -fixpoint. (b) \approx is a $\Sigma \cup \{\varepsilon\}$ -fixpoint. (c) \approx is a Σ^* -fixpoint.

Sketch of Proof: (a) and (b) can be easily shown from their inductive definitions, and the fact that the processes are *finite*; (c) is actually implied by the exposition of [M1] for finite state processes (also see [Sm]). Note that for infinite state processes, the fixpoint property (a) of \approx no longer holds (see [Sn]). \square

Based on this Lemma and the state of the art of CCS, we have a number of interesting propositions that relate the various notions of equivalence. Also, in the standard model whenever p, q denote start states of finite state automata (f.s.a.'s) accepting the same language, we will say $L(p) = L(q)$.

Proposition 1: For F.S.P. states p, q in the general model,

$$p \approx q \text{ iff } p \simeq q. \quad \square$$

Proposition 2: For F.S.P. states p, q in the standard model, (i.e. p, q represent start states of n.f.s.a.'s)

$$p \approx_1 q \text{ iff } L(p) = L(q). \quad \square$$

Proposition 3: For F.S.P. states p, q in the deterministic model, the following are equivalent:

- (a) $p \approx q$ (or $p \simeq q$, or $p \sim q$).
- (b) for some $k \geq 1$, $p \approx_k q$.
- (c) When the model is also standard, (i.e., p, q represent d.f.s.a. start states), $L(p) = L(q)$
- (d) When the model is also restricted, $p \equiv q$. \square

Proposition 4 [B]: For F.S.P. states p, q in the restricted model,

$$p \approx_2 q \text{ implies } p \equiv q \text{ which implies } p \approx_1 q. \quad \square$$

In this paper, we study the complexity of testing whether two states are observation equivalent (in the general model), congruent (in the observable model), or failure equivalent (in the restricted model).

We should note that even in the r.o.u. model, the equivalence notions \approx_k , \equiv , and \approx (or \simeq , or \sim) are different. This is illustrated by the examples of Figure 2.

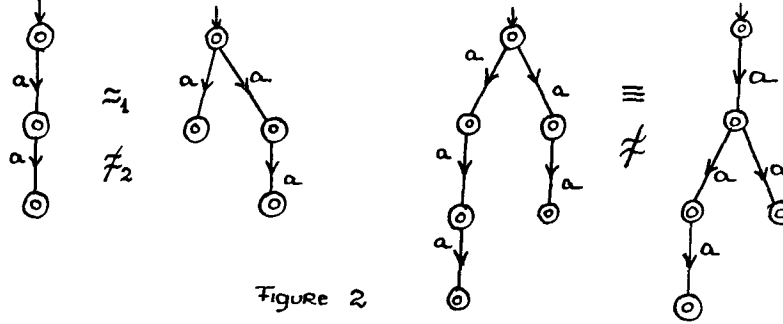


Figure 2

2.3 Regular Expressions for Languages vs. Star Expressions for CCS

The theory of *CCS expressions* is developed in [M2]. Their *syntax* is very similar to that of *regular expressions* for languages. (The notation is slightly different and the introduction of variable symbols makes them more general.) The semantics of a CCS expression is no longer a language but a set of observable F.S.P.'s with congruent start states (see Figure 3a).

A particularly interesting class of these expressions are the *star expressions*. These use the familiar \cup , \cdot , $*$ symbols, with new semantics, and provide the link between the two algebraic theories (of CCS and of regular sets). The F.S.P.'s that are the semantics of star expression r are a subset of the n.f.s.a.'s that accept the language denoted by the regular expression r .

Definition 5: The *syntax of star expressions over Σ* is the same as that of regular expressions over Σ . The *semantics of star expression r* is the set of observable and standard F.S.P.'s whose start states are congruent to p_0 , where $\langle K, p_0, \Delta, E \rangle$ is the *representative F.S.P.* of r . The representative F.S.P. of r is the n.f.s.a. (without empty moves) constructed inductively as in Figure 3b. \square

Intuitively, the meaning of a star expression r is the set of F.S.P.'s whose start states are "equivalent" to the start state of the n.f.s.a. constructed inductively in showing that the language denoted by regular expression r is accepted by some n.f.s.a.. For a precise statement of CCS semantics, see [M2], which also shows that using congruence as the notion of equivalence makes the semantics independent of the representative F.S.P. chosen.

The **CCS-EQUIVALENCE** problem is: "Given two CCS expressions, do they have the same meaning?". This parallels the **EQUIVALENCE** problem for regular expressions [HU, StM]. Using Definition 5 we have:

Lemma 2: Given two star expressions r, r' of length n over a fixed alphabet Σ , their representative (observable and standard) F.S.P.'s have $O(n)$ states and $O(n^2)$ transitions, and can be constructed in $O(n^2)$ time. \square

The above lemma is also true for CCS expressions and observable (though not necessarily standard) F.S.P.'s. Therefore, the CCS-EQUIVALENCE problem is in essence one of testing F.S.P.'s, of similar size, for "equivalence", which is the combinatorial problem examined in this paper.

Finally, let us briefly mention a number of interesting connections between star (CCS) expressions and regular expressions from [M2].

- (1) Every observable F.S.P. is a member of some set of observable F.S.P.'s that are the meaning of a CCS expression, (the variables of Def. 1 are used here).
- (2) There is a complete inference system for identities of CCS expressions paralleling that of Salomaa for regular expressions [Sa].
- (3) The significant algebraic properties that regular expressions have and star expressions lack are the following two identities:

$$r \cdot (r' \cup r'') = r \cdot r' \cup r \cdot r'' \quad \text{and} \quad r \cdot \phi = \phi.$$

For expressions we have limited ourselves to observable F.S.P.'s (no τ -transitions). This is because τ -transitions greatly complicate the algebraic theory, although there still are elegant results [B], for the finite tree model.

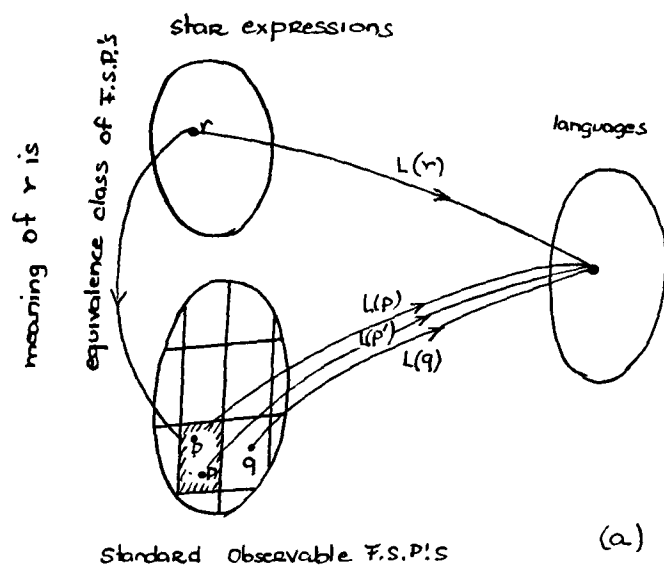

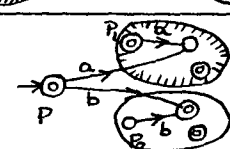
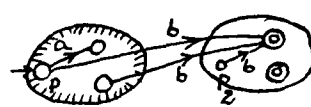
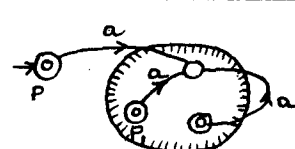


Figure 3

| Syntax | Semantics (the representative F.S.P.) |
|----------------------------------|---|
| \emptyset | $\rightarrow \odot p$ |
| $a \text{ (in } \Sigma \text{)}$ | $\rightarrow \odot_p \xrightarrow{a} \odot_q$ |
| Let r_1, r_2 have semantics |  then |
| $(r_1 \cup r_2)$ |  |
| $(r_1 \cdot r_2)$ |  |
| $(r_1)^*$ |  |

(b)

3. Efficient Algorithms for Testing Congruence

The problem of language equivalence of two finite state automata of size n (i.e. whether they accept the same language) has received a great deal of attention in the literature. For d.f.s.a.'s there is a $O(n G(n))$ algorithm that uses UNION-FIND [AHU, §4.8], and for n.f.s.a.'s the problem has been shown to be PSPACE-complete [StM]. Also, the problem of minimizing the states of a d.f.s.a. of size n has an elegant $O(n \log n)$ solution, and is related to a combinatorial partitioning problem [AHU §4.13, H].

For testing congruence of states of deterministic F.S.P.'s, the above techniques for d.f.s.a.'s are directly applicable (see also Proposition 3). For the larger class of observable F.S.P.'s, congruence of states can still be tested efficiently! This is one more justification for choosing congruence as the appropriate notion of equivalence in CCS. In this case, unfortunately, the UNION-FIND technique does not lead to an efficient algorithm because of possible multiple transitions for one symbol of the alphabet. However, we can show that congruence of states can be tested by solving the following partitioning problem, which is also of independent interest. A *partition* of a set S consists of disjoint nonempty subsets of S called *blocks*, whose union is S .

PARTITIONING

Input: A set S , a partition of S into disjoint blocks $\pi = \{B_1, B_2, \dots, B_p\}$, and a function $f: S \rightarrow 2^S$.

Output: A partition of S into disjoint blocks $\pi' = \{E_1, E_2, \dots, E_q\}$, such that:

1. π' is *consistent* with π (i.e. each E_i is a subset of some B_j).
2. For a, b in block E_i and any block E_j $f(a) \cap E_j \neq \emptyset$ iff $f(b) \cap E_j \neq \emptyset$.
3. π' is the coarsest such partition (i.e. has the fewest blocks).

Obviously, f can be represented as a directed graph with node set S and arcs (i, j) iff j is in $f(i)$. The size of an instance of PARTITIONING is (n, m) , where we denote $|S|$ as n and the number of arcs in the corresponding graph as m . In the deterministic case, we have $f: S \rightarrow S$ and $m = n$.

Intuitively, the initial partition π is refined into π' (in the coarsest fashion possible) so that f induces a mapping from blocks of π' to sets of blocks of π .

Lemma 3: Let p, q be the start states of two observable F.S.P.'s over the fixed sets of actions Σ and variables V . Let these F.S.P.'s have a total of n states and a total of m transitions. We can test whether $p \sim q$ by linear-time reducing it to a PARTITIONING problem of size at most (n, m) .

Sketch of Proof: Using the definition of congruence (Definition 3) and the fact that it is a $\Sigma \cup \{\varepsilon\}$ -fixpoint (Lemma 1), we start from an initial partition based on the extensions of the F.S.P. states and treat each individual symbol of Σ separately. \square

Note, that the PARTITIONING problem is *different* from that of minimizing the states of a d.f.s.a., because having more than one symbols in Σ is different from having many transitions for each symbol, we therefore have to generalize [H].

An obvious solution to the PARTITIONING problem is, starting from π , refine the blocks of the partition by the following method. Let B_i be a block. Examine $f(a) \subseteq S$, for each a in B_i . We can think of $f(a)$ as denoting a set of blocks (those blocks such that each one contains some element of $f(a)$). Now we partition B_i so that two elements a and b are put in the same block if and only if $f(a)$ and $f(b)$ denote the same set of blocks. We will refer to this method as the *naive method*.

Lemma 4: The *naive method* solves an instance of the PARTITIONING problem of size n and m , and can be implemented in $O(nm)$ time.

Sketch of Proof: It is easy to see that the method described above gives the correct output partition. However, the $O(nm)$ implementation is slightly tricky. We perform $O(n)$ iterations since there can be at most n blocks. In each iteration the lexicographic sorting method from [AHU] is used twice and takes $O(n+m)$ time. Also, simple examples show that this bound is tight for our implementation. \square

For the important case of bounded fanout, we can improve upon the naive method by generalizing the divide-and-conquer method of [AHU]. We say that we have *bounded fanout* if for all a in S , $|f(a)| \leq c$, for some constant c . This case corresponds to F.S.P.'s that have at most c transitions out of any state, for each symbol of the action alphabet. In what follows, we describe the intuition behind this algorithm for fanout bounded by constant c .

For each block $B \subseteq S$, define the following c sets:

$f_i^{-1}(B) = \{b \mid f(b) \cap B \neq \emptyset \text{ and there are exactly } i \text{ other distinct blocks } B' \text{ such that } f(b) \cap B' \neq \emptyset, 0 \leq i \leq c-1\}$.

For example, if $|f(a)| = 2$, for all a in S , given B we have $f_0^{-1}(B) = \{b \mid |f(b) \cap B| = 2\}$ and $f_1^{-1}(B) = \{b \mid |f(b) \cap B| = 1\}$. In the deterministic case of [AHU], we have $|f(a)| = 1$, for all a in S , and only $f_0^{-1}(B)$. Note that all the $f_i^{-1}(B)$'s, $0 \leq i \leq c-1$, are clearly distinct sets. Now, instead of partitioning a block B by the values of $f(a)$ for a in B , we partition with respect to B those blocks B_i that do not lie entirely inside one of the $f_i^{-1}(B)$ or that lie outside of all of them (see Figures 4a and 4b). Thus, B_i is partitioned into at most $c+1$ blocks:

$$B_i^i = \{b \mid b \in B_i \text{ and } b \in f_i^{-1}(B)\},$$

$$0 \leq i \leq c-1, \text{ and}$$

$$B_i^c = B_i - \bigcup_i B_i^i.$$

Once we have partitioned with respect to B , we need not partition again with respect to B unless B itself is split. However, B itself may be split into at most $c+1$ pieces $B^0, B^1, B^2, \dots, B^c$. As in [AHU], we need not partition further with respect to all of $B^0, B^1, B^2, \dots, B^c$; we need only further partition with respect to the smallest c of them. However, now, the proof of correctness is slightly more complex. For $c = 2$, the justification for this divide-and-conquer is given in Figure 4c: by partitioning with respect to any two of B^0, B^1, B^2 , we get the other one for free.

```

Input:  $S, f, \pi$ 
begin
BLOCKS :=  $\pi$ ;
WAITING :=  $\pi$ ;
while WAITING not empty do
begin select and delete a block  $B$  from WAITING;
compute  $f_i^{-1}(B)$  for  $0 \leq i \leq c-1$ ;
for each  $B_i$  in BLOCKS that intersects some
 $f_i^{-1}(B)$  and is not its subset do
begin
delete  $B_i$  from BLOCKS;
partition  $B_i$  creating  $k$  new blocks in BLOCKS so that
each new block is entirely in some  $f_i^{-1}(B)$  or outside all of them;
if  $B_i$  is in WAITING then delete it from WAITING
and add all new blocks to WAITING;
else add the  $k-1$  smallest blocks to WAITING;
end
end
return  $\pi' :=$  BLOCKS;
end

```

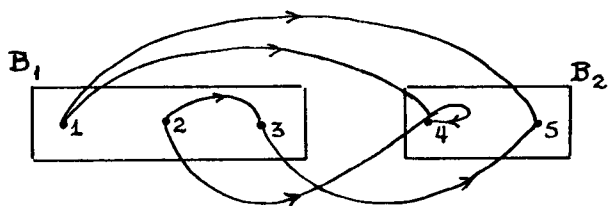
Algorithm ALG

Based on this divide-and-conquer, and an analysis of implementation and running time similar to [AHU], we have the following algorithm, ALG, and Theorem 1.

Theorem 1: The bounded fanout PARTITIONING problem (i.e. for all a in S , $|f(a)| \leq c$) is correctly solved by algorithm ALG in time $O(c^2 n \log n)$. \square

As a result we have:

Corollary 1.1: Let p, q be two states of observable F.S.P.'s, where the F.S.P.'s have n states and m transitions. Then $p \sim q$ can be decided in $O(nm)$ time and, in the case of F.S.P.'s with at most c transitions out of every state on any action, in $O(c^2 n \log n)$ time. \square



(a) S, π and f

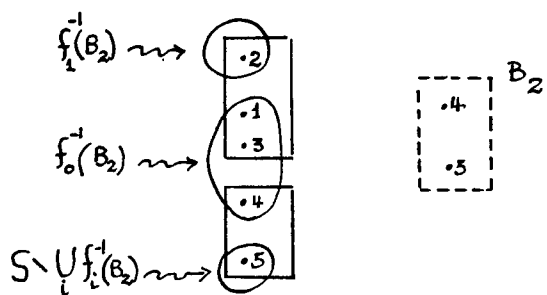
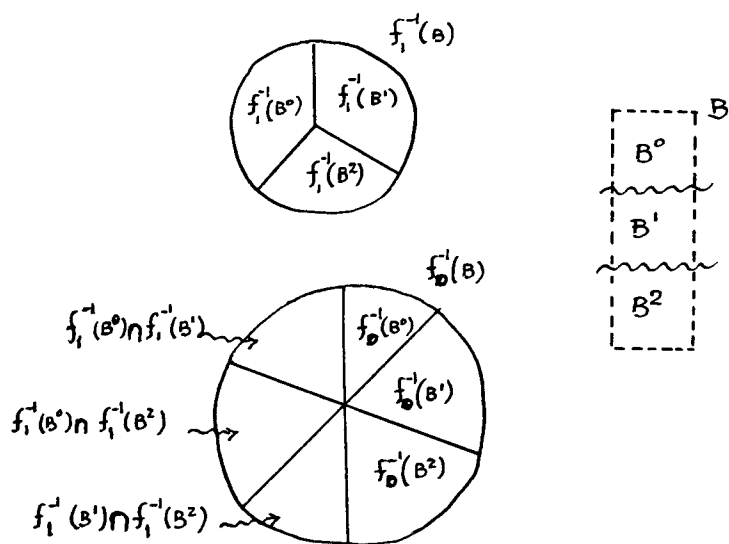


Figure 4

(b) partitioning with respect to B_2



(c) partitioning with respect to B , and splitting B .

4. The Complexity of Observation Equivalence

In this section, we examine the complexity of testing for observation equivalence, a notion used for states of F.S.P.'s from all models described in Section 2.1. The upper bounds presented in this section hold even for F.S.P.'s of the general kind, and the lower bounds even for F.S.P.'s that are restricted and observable (in some cases also r.o.u.).

Theorem 2: Let p, q be F.S.P. states and assume that the F.S.P.'s to which these states belong have a total of n states.

- a) In the general model, whether $p \approx q$ can be decided in $O(n^3)$ time.
- b) In the restricted and observable model, whether $p \approx_k q$ for some fixed $k \geq 1$ is PSPACE-complete.
- c) In the r.o.u. model, whether $p \approx_k q$ for some fixed $k \geq 2$ is NP-complete, and decidable in linear time for $k = 1$.

Sketch of Proof:

(a) We know from Proposition 1 that $p \approx q$ iff $p \simeq q$. By the definition of \simeq (Definition 2), we see that we have a problem that is very similar to that of Section 3; our only additional consideration is τ -transitions. A version of Lemma 3 still holds as long as we take paths of τ -transitions into account. Using a "transitive closure" type operation, we replace the transitions in the F.S.P. with arcs (p, q) labelled with σ , whenever $p \xRightarrow{\tau} q$ and $\sigma \in \Sigma \cup \{\epsilon\}$. Note that there are at most $O(n^2)$ such new arcs. Now we can apply directly the technique of the previous section (i.e. the naive method) to get an $O(n^3)$ algorithm.

(b) Let p be a state of a standard F.S.P. (a n.f.s.a.) and let $L(p)$ be the language accepted by this n.f.s.a. with start state p . We know from Proposition 2 that $p \approx_1 q$ iff $L(p) = L(q)$. Thus, it is PSPACE-complete [StM] to decide \approx_1 in the standard, observable model. As was recently shown [CS], deciding \approx_1 is PSPACE-complete even for the restricted, observable case. Obviously, testing for \approx_0 is trivial.

Membership in PSPACE for \approx_k can be established by a simple reduction to the classical problem of equivalence of two n.f.s.a.'s. By a general reduction trick, we show that deciding \approx_k , for any fixed k , is PSPACE-complete in the restricted, observable case. The technique is the following:

"Given two F.S.P. states p, q , we construct two states p', q' of new F.S.P.'s such that $p \approx_k q$ iff $p' \approx_{k+1} q'$, for $k \geq 1$."

The reduction uses one symbol (the symbol a) from the alphabet Σ , and is illustrated in Figure 5.a. Intuitively:

$$\text{fsp}(p') = \text{fsp}(a) \cdot (\text{fsp}(p) \cup \text{fsp}(q))$$

$$\text{fsp}(q') = \text{fsp}(a) \cdot \text{fsp}(p) \cup \text{fsp}(a) \cdot \text{fsp}(q)$$

Note the relationship between this reduction and (3) in Section 2.3. The correctness of the reduction follows from Definition 1.

As shown in part (a) of this theorem, the hierarchy of problems becomes easy as $k \rightarrow \infty$ (see Figure 5.b).

(c) In the r.o.u. model ($\Sigma = \{a\}$, no τ -transitions, and all states accepting), it is easy to decide $p \approx_1 q$ or $L(p) = L(q)$. This is because languages $L(p), L(q)$ are closed under prefix and are therefore either $\{a\}^*$ or finite initial segments of $\{a\}^*$. However, if p, q are from the *standard* observable model with $\Sigma = \{a\}$, the \approx_1 problem becomes NP-complete [StM]. For $k \geq 2$, we can reduce the r.o.u. \approx_k problem to this NP-complete problem (thus it is in NP). Also this NP-complete problem can be reduced to the r.o.u. \approx_k problem (thus it is NP-hard). This gives us (c). \square

Finally, we would like to point out that in classical complexity theory [StM], there are results for the problem $p \approx_1 q^*$, where q^* is the trivial f.s.a. that accepts Σ^* (see Figure 5.c). Using Definition 2, we can show that testing $p \approx_2 q^*$ is easy. Namely, $p \approx_2 q^*$ iff "every state reachable from p has outgoing transitions for every symbol from Σ ." This is a consequence of the fact that in going from \approx_1 to \approx_2 , we examine all $s \in \Sigma^*$ (whereas in going from \approx_1 to \approx_2 , we examine only $s \in \Sigma$).

5. The Complexity of Failure Equivalence

In this section, we analyze the complexity of testing two states for failure equivalence. We will therefore be working in the restricted model. As described in Section 2, for states p, q we have:

$\text{failures}(p) = \{(s, Z) \mid s \in \Sigma^*, Z \subseteq \Sigma \text{ with the property: there is a state } p', \text{ such that for every } z \text{ in } Z: p \xRightarrow{s} p' \text{ and } p' \not\xRightarrow{z} \text{dead}\}$

and

$$p \equiv q \text{ iff } \text{failures}(p) = \text{failures}(q).$$

The notion of failure equivalence has proven useful for the case of finite trees [B] (with τ -transitions). However, for slightly more complex finite processes, even without τ -transitions, we have:

Theorem 3: Let p, q be states of restricted observable F.S.P.'s. Then, deciding whether $p \equiv q$ is PSPACE-complete, and NP-complete in the r.o.u. model.

Sketch of Proof: Using the definition of failure equivalence, we can reduce it to the classical notion of equality of languages accepted by two n.f.s.a.'s. Thus, in both the restricted observable and the r.o.u. models we can show membership in PSPACE and NP, respectively. So all that remains is to show hardness.

Let p and q be the start states of two restricted observable F.S.P.'s. They can also be viewed as the start states of two n.f.s.a.'s which accept the languages $L(p)$ and $L(q)$, respectively. Note that, as a consequence of the restricted model, the only way a string s is not in $L(p)$ or $L(q)$ is if $p \xRightarrow{s} \text{dead}$ or $q \xRightarrow{s} \text{dead}$.

Given p, q , we will produce two states p', q' such that $L(p) = L(q)$ iff $p' \equiv q'$. (This will give us PSPACE-hardness or NP-hardness.)

Let p be the start state of $\langle K, p, \Delta, E \rangle$, and obtain p' as follows:

- (i) Add a new state p^* (which is also accepting) to K to obtain $K' = K \cup \{p^*\}$.
- (ii) Augment Δ when, $p_i \in K$, $\sigma \in \Sigma$, by: $p^* \in \Delta(p_i, \sigma)$. (Note that p^* is a trap state with incoming arcs from all other states.)

Now in the augmented F.S.P. rename p , the start state, as p' . Similarly, obtain q' from q .

Obviously, we have, $L(p') = L(p) \cup L(p) \cdot \Sigma$, $L(q') = L(q) \cup L(q) \cdot \Sigma$, and

$$\text{failures}(p') = \{(s, \phi) \mid s \in L(p)\} \cup$$

$$\{(s, Z) \mid s \in L(p) \cdot \Sigma \text{ and } Z \subseteq \Sigma\},$$

$$\text{failures}(q') = \{(s, \phi) \mid s \in L(q)\} \cup$$

$$\{(s, Z) \mid s \in L(q) \cdot \Sigma \text{ and } Z \subseteq \Sigma\}.$$

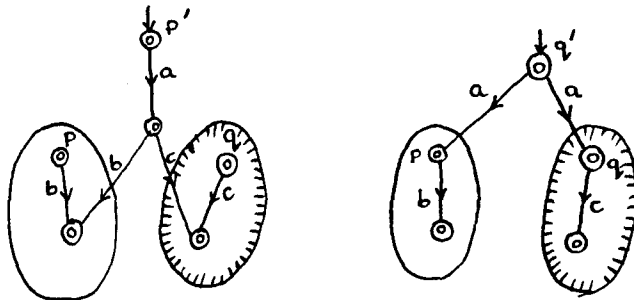
Hence, $p' \equiv q'$ iff

$$(a) \quad L(p) \cdot \Sigma = L(q) \cdot \Sigma$$

$$(b) \quad L(p) \cup L(p) \cdot \Sigma = L(q) \cup L(q) \cdot \Sigma$$

It is easy to see that $L(p) = L(q)$ implies $p' \equiv q'$. If $p' \equiv q'$ then $L(p) \cdot \Sigma = L(q) \cdot \Sigma$ and, because in the restricted model $L(p)$ and $L(q)$ are prefix-closed, we have $L(p) = L(q)$.

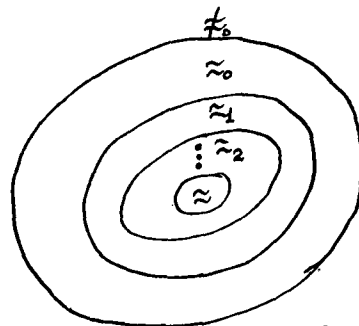
This completes the reduction. \square



(a) the reduction



(c) the trivial F.S.P.



(b) hierarchy for observable F.S.P.'s.

Figure 5

6. Discussion and Open Problems

We have investigated the complexity of three equivalence notions that are central to the definition of CCS semantics. We have tried to draw a close analogy between finite state processes and expressions in CCS on one hand, and finite state automata and regular expressions on the other. We believe that in CCS, an algebraic model for distributed computation, many of the classical problems are cast in a new light. An example is the *star height* question about star expressions raised in [M2]. We would like to point out two open problems which we believe are both interesting and important.

(a) PARTITIONING: We conjecture that the PARTITIONING problem of size (n, m) has an $O(m \log n)$ solution. This does not follow from the algorithm of Section 3, which handles the bounded fanout case.

(b) CCS-EQUIVALENCE: For the star expressions defined in Section 2.3, the CCS-EQUIVALENCE problem is essentially that of testing finite state processes (of size comparable to that of the expressions) for congruence of start states. CCS, being a calculus, provides a number of other algebraic operators besides \cup , \cdot , $*$. Therefore, as we have *extended regular expressions* in the classical theory [StM], we have *extended star expressions* in CCS. Since the semantics of CCS is in terms of sets of processes rather than strings, an operator such as complement ($-$) would make little sense in the new context. However, operators such as *composition* [M1] or intersection can be given new semantics. (Composition is one of the main distributed features of CCS.) The new semantics are, predictably, in terms of a "direct product of states" construction. In the spirit of Definition 5, the representative process of the whole is the result of taking the direct product of the representative processes of the parts. With extended star expressions, the CCS-EQUIVALENCE problem acquires new interest. Extended star expressions are succinct programs with large representative finite state processes, because of possible nesting of the new operators. So perhaps the CCS-EQUIVALENCE problem becomes hard, as do its counterparts in [StM].

We would like to end this discussion on a note of optimism. For regular expressions, the MEMBER problem (i.e., is string s in the language denoted by regular expression r ?) is different from the EQUIVALENCE problem and solvable efficiently by dynamic programming

[HU, §4.5]. For extended star expressions, the distinction between CCS-MEMBER (i.e., is state p in the equivalence class denoted by CCS-expression r) and CCS-EQUIVALENCE is much weaker. Therefore, we conjecture that both PARTITIONING, in general, and CCS-EQUIVALENCE, for extended star expressions, have elegant and efficient solutions.

Acknowledgements: We would like to thank Gordon Plotkin, Ashok Chandra Larry Stockmeyer and Alessandro Giacalone for many helpful discussions, as well as Ashfaq Munshi for his comments on the presentation of these results. The research of the first author was supported by NSF grant MCS-8210830.

References

- [AHU] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).
- [B] S.D. Brookes, "On the Relationship of CCS and CSP", technical report, Department of Computer Science, Carnegie-Mellon University, (1982).
- [CS] A.K. Chandra, L.J. Stockmeyer, private communication (July 1982).
- [H] J. Hopcroft, "An $n \log n$ Algorithm for Minimizing States in a Finite Automaton", in *Theory of Machines and Computations*, pp. 189-196, Eds. Z. Kohavi and A. Paz, Academic Press, New York (1971).
- [HBR] C.A.R. Hoare, S.D. Brookes, A.W. Roscoe, "A Theory of Communicating Sequential Processes", Technical Monograph PRG-16, Oxford University Computing Laboratory, Programming Research Group, Oxford, England (May 1981).
- [HU] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley (1979).
- [M1] R. Milner, "A Calculus of Communicating Systems", *Lecture Notes in Computer Science* 92, Springer-Verlag (1980).
- [M2] R. Milner, "A Complete Inference System for a Class of Regular Behaviors", Department of Computer Science, University of Edinburgh, Internal Report CSR-111-82 (April 1982).

- [M3] R. Milner, "Four Combinators for Concurrency", *Proc. ACM Symp. on Principles of Distributed Computing*, Ottawa, Canada (Aug. 1982).
- [Sa] A. Salomaa, "Two Complete Axiom Systems for the Algebra of Regular Events", *Journal ACM*, Vol. 13, No. 1, pp. 158-169, (1966).
- [Sm] S.A. Smolka, Ph.D. Dissertation
- [Sn] M.T. Sanderson, "Proof Techniques for CCS", Department of Computer Science, University of Edinburgh, Internal Report CST-19-82 (Nov. 1982).
- [StM] L.J. Stockmeyer, A.R. Meyer, "Word Problems Requiring Exponential Time", *Proc. 5th Ann. ACM Symp. on Theory of Computing*, pp. 1-9, Austin, TX (April 1973).