On the Analysis of Cooperation and Antagonism in Networks of Communicating Processes

Paris C. Kanellakis[†]
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02138
U.S.A.

Scott A. Smolka[‡]
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794
U.S.A.

Abstract

We propose a new method for the analysis of cooperative and antagonistic properties of communicating finite state processes (FSPs). This algebraic technique is based on a composition operator and the notion of "possibility equivalence" among FSPs. We demonstrate its utility by showing that potential blocking, lockout, and termination can be efficiently decided for loosely connected networks of tree FSPs. If not all acyclic FSPs are trees, then the cooperative properties become NP-complete and the antagonistic ones PSPACE-complete. For tightly coupled networks of tree FSPs, we also have NP-hardness. For the considerably harder cyclic case, we provide a natural extension of the method as well as a subcase reducible to integer programming with a constant number of variables.

1. Introduction

There has been a great deal of interest in recent years in algebraic approaches to the specification and verification of concurrent systems [e.g. M, Br, HBR, H]. The motivation behind these approaches is to simplify the analysis of potential termination, blocking, deadlock, lockout, liveness and other properties of communicating processes. These static analysis tasks are central to both the areas of concurrent programming and network protocol validation [e.g. BZ, CES, H, L, OL, RT, S, T]. Unfortunately, if no use of the particular problem structure is made, the general computational problems are usually intractable [L, T, GJ, RT].

We demonstrate the use of an algebraic method which indeed exploits the particular structure of the given problem. We present efficient techniques for the analysis of many properties of a system of communicating finite state processes. These properties refer both to cooperating and to antagonistic processes.

Our model is a network of finite state processes. A non-empty transition (action) of a process represents an instruction for this process to exchange a "handshake" with another specified process. An empty transition (τ-move) represents an internal change of the process undetectable by the outside world.

Two processes P, Q can be *composed* into a combined process $P \,||\, Q$, which represents their behavior with respect to the rest of the world and partially hides the interaction between them from the rest of the world. The composition of all processes in a network results in one *global* process with only τ-moves. Analysis of the reachability properties of the global process is a standard, albeit inefficient, way of studying the behavior of the network.

It is reasonable to assume that the global process will change state if possible. This *continuity rule* (such as the one in [L]) forces system evolution to take place. Let $N = \{P_1, P_2, \cdots, P_m\}$ be our network of processes, $P = P_1$ the process in the network we wish to study, and $Q = P_2 \,||\, P_3 \,||\, \cdots \,||\, P_m$ the rest of the network which P views as a process. (Note that intentionally $||$ is devised to be associative and commutative.)

Process P, due to branching actions in its state diagram and to τ-moves, can make choices that affect the behavior of the system. *How must P schedule its choices so that it evolves in a "desirable fashion"?* By desirable fashion we mean "reaches an accept state" for an *acyclic* (finite) process and "never stops" for a *cyclic* (infinite) process. We study three related questions which appear to be fundamental:

(a) Can P make its choices randomly? That is, is it guaranteed to evolve desirably no matter what its choices are? We call this property *unavoidable success*. Its negation is sometimes known as *potential blocking*.

[†]On leave from Brown University. Supported by NSF grant DCR-8302391, ONR-DARPA grant N00014-83-K-0146, and by the Office of Army Research under Contract DAAG29-84-K-0058.

[‡]Supported by NSF grant DCR-8319966.

(b) Can P make its choices intelligently and locally? That is, can it schedule its choices based only on its partial information about the system and still evolve desirably no matter how antagonistic the other processes are? We call this property *success in adversity*. This is a simple and yet complete version of the *no-lockout* problem posed in [L].

(c) It is even possible for P to evolve desirably? That is, can P and the rest of the network collaborate to achieve the desired evolution of P? We call this property *success with collaboration*. It is the weakest of the three notions and does not correspond to any realistic scheduling strategy. It is sometimes known as *potential termination*.

A reasonable assumption about many networks N is that processes are "loosely interconnected". That is, the graph representing which pairs of processes can handshake has a low degree of connectivity. This C_N could be a tree, a ring, or more generally what we call a *k-tree*. The sparsity of C_N is the problem structure we would like to take advantage of using our algebraic method. For simplicity, let us think of C_N as a tree with P, the *distinguished* process, determining its center.

Two critical choices in our method of analyzing P's behavior in context Q are:

(1) the choice of composition operator,

(2) the choice of equivalence notion among processes.

The composition operator must be associative, commutative and have the following property: If in a composition operation $R_1 || R_2$ we replace R_2 with an equivalent R_2', the new composition $R_1 || R_2'$ is equivalent to $R_1 || R_2$. Also, the equivalence notion must be sufficient to characterize unavoidable success, success in adversity, and success with collaboration.

We define the appropriate operator $||$ for acyclic processes in Section 2.2, and a technical variant of it for cyclic processes in Section 4. The right notion of equivalence is a refinement of the [HBR] *failure equivalence*, and is based on the *possibilities Poss(R)* of a process R. A possibility (s, Z) of a process is a pair consisting of (1) a string of transitions s bringing the process to a state it cannot change silently and (2) a set Z of all possible next transitions at this state.

Using $||$, we combine processes from the leaves of C_N towards the center, always making sure that possibilities are preserved. We believe that this is *the* natural organized analysis of our generally hard combinatorial questions. We show that $||$ and $Poss(\cdot)$ are the appropriate choices because of the efficiency in solving the finite case, and the small modification necessary to extend the finite case to the harder infinite case.

We first show that if we have straight-line processes all the questions of success become one that can be answered easily by a causality argument (Proposition 1). If we have small tree processes interconnected in a tight fashion, unavoidable success is co-NP-complete and success with collaboration is NP-complete (Theorem 1). These are improvements on bounds derived in [T] for a different process model. For success in adversity of loosely interconnected acyclic processes, where the distinguished process describes an exponential number of paths, we show PSPACE-completeness (Theorem 2). This demonstrates the harder game nature of antagonism versus collaboration (which is in NP) in a simpler fashion than was done in [L]. The NP-hardness bounds of the acyclic case become PSPACE-hardness bounds in the cyclic case. Exponential bounds, similar to [L], can be shown for success in adversity (Proposition 2).

These intractability results provide the setting for our main concern, which is the choice of $||$ and $Poss(\cdot)$ and their use in showing that all three notions of success can be efficiently decided for loosely interconnected networks of tree processes (Theorem 3). Lemmas 2 (for acyclic processes) and 2′ (for cyclic processes) are critical to the analysis. Lemmas 3,4,5 express the notions of success in terms of possibilities and analyze success in adversity as a game of partial information [R] between P and Q. Theorem 3 demonstrates that possibilities provide an efficient data structure in this case.

Their are two indications that our approach can provide a practical heuristic for the harder case of loosely interconnected cyclic processes. One is the simple modification of composition (Section 4), which preserves most of the algebraic properties of the acyclic case. The notion of possibilities is not modified at all, however it can no longer provide an efficient data structure in the worst case [KS]. The second indication is that using the weaker "language equivalence" notion, our method efficiently solves a special case of the success-with-collaboration question; i.e. in tree networks of small processes using only one type of handshake. Despites this problem's restricted nature, we have had to use the powerful technique in [Le] to maintain efficiency.

Section 2 contains our model, Section 3 the analysis of the acyclic case, and Section 4 the extensions and analysis for the cyclic case.

2. The Model

2.1. Networks of Processes

The finite state process, the basic building block of our model, closely resembles the nondeterministic finite state automaton (NFA) of the classical theory of computation.

Definition 1: A *Finite State Process* (FSP) is a quadruple $<K, p, \Sigma, \Delta>$, where:

1) K is a finite set of *states*;

2) $p \in K$ is the *start state*;

3) Σ is a set of symbols called *actions*, and $\tau \notin \Sigma$ is a special symbol called the *unobservable action*;

4) $\Delta \subseteq K \times (\Sigma \cup \{\tau\}) \times K$ is a relation called the *transition relation*.

We also assume that every state in K is reachable from the start state p using some sequence of transactions. |

An FSP can be represented as a directed graph, whose nodes are the states and, for each (p, λ, p') in Δ, there is an arc from state p to state p' labeled by λ. This syntax is identical to that of NFAs with empty transitions when all states are accepting states. We will use capital letters $P, Q, \cdots$ to denote FSPs and small letters $p, q, \cdots$ to denote states.

Let $s \in \Sigma^*$ and p, p' be states of an FSP. If $s = \epsilon$ (i.e. s is the empty string), we say that $p \stackrel{s}{\Rightarrow} p'$ when there is a sequence of k arcs in the graph of the FSP from p to p' with labels τ^k, $k \geq 0$. (Obviously always $p \stackrel{\epsilon}{\Rightarrow} p$.) If $s = s_1 s_2 \cdots s_l$, $s_i \in \Sigma$, $1 \leq i \leq l$, we say that $p \stackrel{s}{\Rightarrow} p'$, when there is a sequence of $k_0 + k_1 + \cdots + k_l + l$ arcs in the graph of the FSP from p to p' with labels $\tau^{k_0} s_1 \tau^{k_1} s_2 \cdots s_l \tau^{k_l}$, $k_0, k_1, \cdots, k_l \geq 0$. We distinguish between τ and ϵ because the unobservable action τ plays a special role in distributed computation [M].

Since there is no distinction between "accept" and "reject" states, the only feature that distinguishes FSP states is the absence of certain transitions. Let p be a state and $s \in \Sigma^*$. We say that $p \stackrel{s}{\Rightarrow} \mathit{dead}$, when there is no p' such that $p \stackrel{s}{\Rightarrow} p'$. This concept is formalized in [HBR] as the *failures* of state p (or $\mathit{Fail}(p)$).

$$\mathit{Fail}(p) = \{(s, Z) \mid s \in \Sigma^*, Z \subseteq \Sigma \text{ such that: } \exists p' \ (p \stackrel{s}{\Rightarrow} p') \text{ and } (\forall z \in Z, \ p' \stackrel{z}{\Rightarrow} \mathit{dead})\}$$

Let $<K, p, \Sigma, \Delta>$ be an FSP and let G be the directed graph representing it. If G is a path we have a *linear* FSP; if it is a tree (rooted at p) we have a *tree* FSP; and if it is a directed acyclic graph with single root p, we have an *acyclic* FSP. A state of the FSP with no transitions leaving it is called a *leaf*.

The meaning of the actions of an FSP is messages exchanged with other FSPs. For example, if $x \in \Sigma_1 \cap \Sigma_2$ then x is a message that FSP_1 could exchange with FSP_2. As will be clear from the definition of composition (given in the following subsection), the message exchange is in the form of a "handshake" between the two processes. Intuitively, no distinction is made between *send* and *receive*. A message can only be exchanged between two processes, i.e. communication is point-to-point. The meaning of a τ is a step inside the FSP invisible to the outside world.

Definition 2: A *network N of processes* is a set of m FSPs, $N = \{P_1, P_2, \cdots, P_m\}$, where we let P_i denote $<K_i, p_i, \Sigma_i, \Delta_i>$, and

1) the K_i's are distinct sets of states, $1 \leq i \leq m$;
2) each $x \in \bigcup_i \Sigma_i$ belongs to *exactly two* process sets of actions. |

Therefore a network N is a closed system of communicating processes. Since each action symbol belongs to exactly two processes, we can describe the potential to communicate using a labeled undirected graph C_N. The nodes of C_N correspond to the processes in N and there is an edge $\{i, j\}$ between nodes i and j iff $\Sigma_i \cap \Sigma_j \neq \emptyset$. The label of the edge $\{i, j\}$ is $\Sigma_i \cap \Sigma_j$ (i.e. process p_i can communicate with process p_j using any $x \in \Sigma_i \cap \Sigma_j$). If C_N is a tree (ring), we say that *network N is a tree (ring)*.

Let N be a network of processes with $C_N = (V, E)$, and π a *given* partition $(V_1, V_2, \cdots, V_l)$ of V into disjoint sets. We will call N a *k-tree* if:

(a) $|V_i| \leq k$, $(i = 1,2, \cdots, l)$
(b) the graph on nodes $\{1, 2, \cdots, l\}$ and edges $\{(i, j) \mid$ where E contains an edge with endpoints in V_i and $V_j\}$ is a tree.

Note that a tree network is a 1-tree, a ring network a 2-tree, and if the largest biconnected component of C_N has size k we have a k-tree.

2.2. Process Composition: Algebraic Properties

We can now describe the interaction of processes in a network using the algebraic operation of *composition* ($||$).

Definition 3: Let N be a network of FSPs and $P_1 = <K_1, p_1, \Sigma_1, \Delta_1>$, $P_2 = <K_2, p_2, \Sigma_2, \Delta_2>$ two distinct processes in N. Let:

$$P_1 \times P_2 = <K_1 \times K_2, (p_1, p_2), (\Sigma_1 \cup \Sigma_2), \Delta>$$

where the new transition relation Δ is defined as follows:

if $(q_1, \lambda, r_1) \in \Delta_1$ and $(q_2, \mu, r_2) \in \Delta_2$ then

if $\lambda \in (\Sigma_1 \cup \{\tau\}) - \Sigma_2$ then $((q_1, q_2), \lambda, (r_1, q_2)) \in \Delta$

if $\mu \in (\Sigma_2 \cup \{\tau\}) - \Sigma_1$ then $((q_1, q_2), \mu, (q_1, r_2)) \in \Delta$

if $\lambda = \mu \in \Sigma_1 \cap \Sigma_2$ then $((q_1, q_2), \lambda, (r_1, r_2)) \in \Delta$ |

The FSP $P_1 \cap P_2$ is $P_1 \times P_2$ restricted to states reachable from the start state (p_1, p_2).

The FSP $P_1 || P_2$ is $P_1 \cap P_2$ with all actions in $\Sigma_1 \cap \Sigma_2$ replaced by the unobservable action τ. $P_1 || P_2$ is the *composition* of P_1 and P_2.

Let N be the network of FSPs $\{P_1, P_2, \cdots, P_m\}$. The transitions of $P_1 \times P_2$ are either the moves of P_1 with respect to $P_3, \cdots, P_m$, or moves of P_2 with respect to $P_3, \cdots, P_m$, or simultaneous moves occur-

ring whenever P_1 and P_2 can "handshake". $P_1 \cap P_2$ restricts $P_1 \times P_2$ to the relevant moves, i.e. those reachable from the start state. In $P_1 \cap P_2$ we have the information of which were the "handshakes" of P_1 and P_2. $P_1 \,||\, P_2$ hides these "handshakes" from the other processes.

Example: In Figure 1a we have an example of a tree network (C_N) of three processes $\{P_1, P_2, P_3\}$, where P_1 is a tree process, P_2 an acyclic process, and P_3 cyclic. $P_1 \times P_2$ is illustrated in Figure 1b, $P_1 \cap P_2$ is the part of $P_1 \times P_2$ reachable from state (1, 1). Finally, the composition $P_1 \,||\, P_2$ is in Figure 1c; this operation transforms C_N into $C_{N'}$.

By performing a sequence of compositions on the processes of a network N, we can produce new processes whose states are, in essence, tuples of states of the processes in N.

Lemma 1: Let P_i, P_j, P_k be three distinct FSPs of a network N. Then $P_i \,||\, P_j = P_j \,||\, P_i$ and $(P_i \,||\, P_j) \,||\, P_k = P_i \,||\, (P_j \,||\, P_k)$. |

Since the states of the processes $P_1, P_2, \cdots, P_m$ of N are distinct, we may, without loss of generality, disregard the order of states in tuples. For example, if $q_i \in K_i$, $q_j \in K_j$ and $q_k \in K_k$, then composite state (q_i, q_j) is the same as (q_j, q_i), and $(q_i, (q_j, q_k))$ is the same as $((q_i, q_j), q_k)$. If we follow this convention in naming composite states, Lemma 1 says that $||$ is *commutative* and *associative*.

A consequence of Lemma 1 is that the process $P_{i_1} \,||\, P_{i_2} \,||\, \cdots \,||\, P_{i_k}$, $1 \le i_1 < i_2 < \cdots < i_k \le m$, is well-defined. A state of this new process is a tuple composed of states from $P_{i_1}, P_{i_2}, \cdots, P_{i_k}$.

The associativity of $||$ is a direct consequence of our assumption that action symbols are shared by exactly two processes in the network, and would otherwise not be true [M].

Possibilities of an acyclic FSP

The operation of composition ($||$) in a network of processes has a number of interesting properties. In particular it matches well with the notion of *possibilities*, which will be a powerful tool in analyzing FSPs.

Definition 4: Let $P = \langle K, p, \Sigma, \Delta \rangle$ be an *acyclic* FSP. The *language* of P and the *possibilities* of P are given as:

$Lang(p) = \{s \mid s \in \Sigma^*$, such that: $\exists q, p \stackrel{s}{\Rightarrow} q\}$.

$Poss(p) = \{(s, Z) \mid s \in \Sigma^*, Z \subseteq \Sigma$, such that:
$\exists q, p \stackrel{s}{\Rightarrow} q$ and
(q has no outgoing τ-moves) and
(q has outgoing set of actions exactly Z) $\}$. |

The possibility (s, Z) is illustrated in Figure 2a. It is a pair consisting of a string s and a set Z of actions. The string s takes the process from the start state p to a state q with no outgoing τ's and with exactly the set of actions in Z as outgoing actions.

Note that $(s, Z) \in Poss(P)$ implies that $s \in Lang(P)$, and $(s, \Sigma - Z) \in Fail(P)$. The sets $Fail(\cdot)$ and $Poss(\cdot)$ are illustrated in Figure 2b, where it is also demonstrated that $Fail(P) = Fail(Q)$ does not imply $Poss(P) = Poss(Q)$.

For acyclic FSPs we have that if $s \in Lang(P)$, there is always at least one $(s, Z) \in Poss(P)$; otherwise, the transition diagram of the FSP would contain a cycle of τ-moves. This last implication is not necessarily true for FSPs with cycles of τ-moves.

For acyclic FSPs it is also easy to show that $Poss(P) = Poss(Q)$ implies that $Fail(P) = Fail(Q)$. Thus the equivalence relation on acyclic FSPs induced by the possibilities is a refinement of the [HBR] failure equivalence.

The set $L = \{s \mid (s, \emptyset) \in Poss(P)\}$ is the language of strings accepted by the leaves of the acyclic process P.

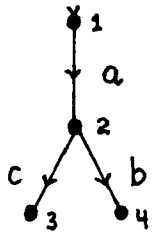
We now show that possibility equivalence is a congruence with respect to composition.

Lemma 2: Let P, P_1, P_2 be acyclic FSPs. Then:

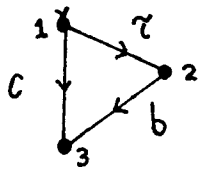
$Poss(P_1) = Poss(P_2)$ implies
$Poss(P \,||\, P_1) = Poss(P \,||\, P_2)$, and
$Lang(P_1) = Lang(P_2)$ implies
$Lang(P \,||\, P_1) = Lang(P \,||\, P_2)$.

Proof: If $Poss(P_1) = Poss(P_2)$ then $Lang(P_1) = Lang(P_2)$. By ignoring (without loss of generality) symbols not present in the transition diagrams of P_1 and P_2, we must have that $\Sigma_1 = \Sigma_2$. Let $(s, Z) \in Poss(P \,||\, P_1)$; we will show that $(s, Z) \in Poss(P \,||\, P_2)$. By symmetry this will suffice.

The string s is over the alphabet $\Sigma \oplus \Sigma_1 = (\Sigma \cup \Sigma_1) - (\Sigma \cap \Sigma_1)$ and Z is a subset of this alphabet. There exists a state in (p', p_1') in $P \,||\, P_1$ such that $(p, p_1) \stackrel{s}{\Rightarrow} (p', p_1')$, (p', p_1') has no outgoing τ-moves, and exactly the z's in Z serve as outgoing actions from this state in $P \,||\, P_1$. By tracing the path that serves as a witness to the fact that (p, p_1) goes to (p', p_1'), we can construct two strings $t \in \Sigma^*$ and $r \in \Sigma_1^*$. These two strings are subsequences of s with additional matching symbols from $\Sigma \cap \Sigma_1$ such that $p \stackrel{t}{\Rightarrow} p'$ and $p_1 \stackrel{r}{\Rightarrow} p_1'$ in FSPs P and P_1, respectively. Therefore the possibilities $(t, Y) \in Poss(P)$ and



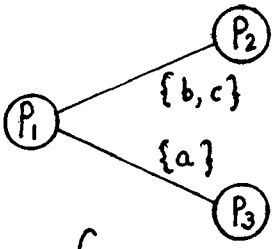
P_1



P_2

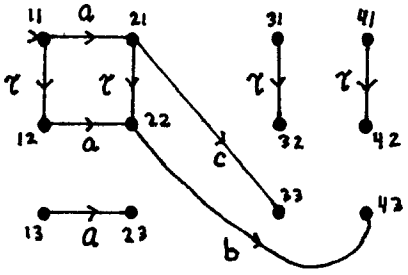
P_3

(a)

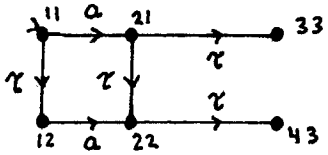


C_N

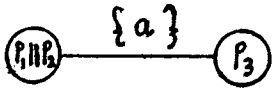
$$Fail(P_2) = (\{c\} \times 2^{\{b,c\}}) \cup (\{b\} \times 2^{\{b,c\}}) \cup (\{\epsilon\} \times 2^{\{c\}})$$



(b) $P_1 \times P_2$



(c) $P_1 \| P_2$

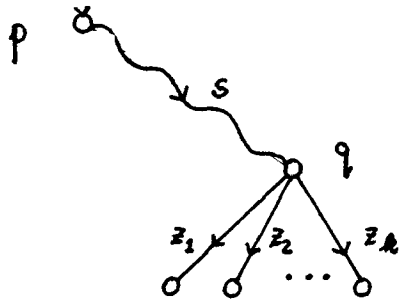


$C_{N'}$

Figure 1

$(r, Y_1) \in Poss(P_1)$ are well-defined. For these possibilities we must have that $Y \cap Y_1 = \emptyset$ and $(Y \cup Y_1) - (\Sigma \cap \Sigma_1) = Z$ (recall $P \,||\, P_1$'s construction and that there are no τ-moves from (p', p_1')). Since $Poss(P_1) = Poss(P_2)$, we have that $(r, Y_1) \in Poss(P_2)$. Thus in $P \,||\, P_2$ we can show that $(p, p_2) \stackrel{s}{\Rightarrow} (p', p_2')$ by simulating the previous decomposition of s into t and r, where p_2' is a state in P_2 witnessing the fact that $(r, Y_1) \in Poss(P_2)$. The transitions out of (p', p_2') in $P \,||\, P_2$ are exactly labeled by the symbols in Z, since $Y \cap Y_1 = \emptyset$, $(Y \cup Y_1) - (\Sigma \cap \Sigma_2) = (Y \cup Y_1) - (\Sigma \cap \Sigma_1) = Z$, and no τ-moves leave p' or p_2' in P and P_2, respectively.

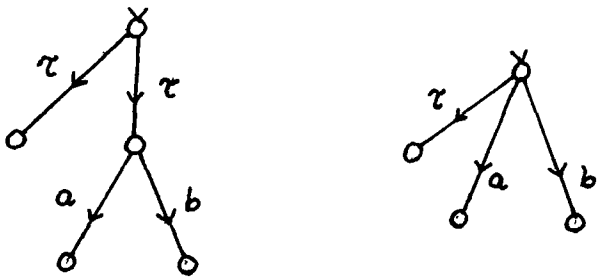
That $Lang(P_1) = Lang(P_2)$ implies $Lang(P \,||\, P_1) = Lang(P \,||\, P_2)$ also follows from an argument similar to the one above. |



no outgoing τ-moves from q

$(s, \{z_1, z_2, \cdots, z_k\}) \in Poss(P)$

(a)



P_1 $\qquad\qquad$ P_2

$$Fail(P_1) = Fail(P_2) = \{\epsilon, a, b\} \times 2^{\{a, b\}}$$

$$Poss(P_2) = \{\epsilon, a, b\} \times \{\emptyset\} \qquad Poss(P_1) = Poss(P_2) \cup \{(\epsilon, \{a, b\})\}$$

(b)

Figure 2

3. Acyclic Processes

Let $N = \{P_1, P_2, \cdots, P_m\}$ be a network of FSPs, let $P = P_1$ be the *distinguished* process, and let Q be its *context* $P_2 || P_3 || \cdots || P_m$. The *global* FSP $G = P_1 || P_2 || \cdots || P_m$, which has only τ-moves, captures all possible state changes in the system.

We wish to study the behavior of P in its context Q; therefore we must assume that the global FSP G changes state if there is a possibility to do so.

Continuity Rule: Let G be in state $(q_1, q_2, \cdots, q_m)$ and let Δ be the set of transitions (i.e. τ-moves to other global states) possible from this state. Unless $\Delta = \emptyset$, one of the transitions in Δ must take place.

In the case of acyclic FSPs, $P = P_1, P_2, \cdots, P_m$, Q, G are all acyclic and G changes state because of the continuity rule until it reaches one of its leaves.

Choices are possible in a network of FSPs. In the simple example of Figure 3, process Q can choose to remain in state 1 in which case, because of the continuity rule, the global FSP must evolve from (1,1) to (2,2) using an "a-handshake" between P and Q. Alternatively, process Q can choose to go to state 3 in which case the global FSP evolves from (1,1) to (3,1).

We will investigate the existence of sequences of choices that lead to "desirable" behavior of G.

3.1. Three Problems of Success

Let $N = \{P_1, P_2, \cdots, P_m\}$ be a network of acyclic FSPs, $P = P_1$ the *distinguished* process, Q its *context* and G the global FSP. We will analyze the possible choices in P so that, under the continuity rule:

(a) P has to reach one of its leaves;

(b) P can reach one of its leaves even if Q is antagonistic;

(c) P can reach one of its leaves if Q collaborates with it.

More formally:

$S_u(P, Q) = true$ **(unavoidable success)**
when
$\forall p', q' . [(p', q') \; leaf\_of \; G] \Rightarrow [p' \; leaf\_of \; P]$.

$S_a(P, Q) = true$ **(success in adversity)**
when
P has a winning strategy in $Game(P, Q)$
(see Figure 4).

$S_c(P, Q) = true$ **(success with collaboration)**
when
$\exists p', q' . [(p', q') \; leaf\_of \; G] \wedge [p' \; leaf\_of \; P]$.

Obviously, $S_u(P, Q) \Rightarrow S_a(P, Q) \Rightarrow S_c(P, Q)$. However, even in very simple cases, such as Figure 3, $S_c(P, Q) \not\Rightarrow S_u(P, Q)$.

We will be using the term *potential blocking* for $\neg S_u(P, Q) = true$. It is easy to see that potential blocking corresponds to:

$$\exists p', q' . [(p', q') \; leaf\_of \; G] \wedge \neg [p' \; leaf\_of \; P].$$

For success in adversity we assume that P, the distinguished process, has no τ-moves. This is to simplify the rules of the *Game* of Figure 4, which captures the evolution of the system. In *Game*, player Q is forced to move if it can (continuity rule) and player P is forced to respond to Q's message. Player Q has a strong adverse role since it knows the global state and selects the next legal action. Player P knows only it local state and can estimate the global state from messages received and Q's structure. Whereas player Q chooses q_{i+1} and a_i (its next local state and the next action), player P only chooses p_{i+1} (its next local state). This is a *game of partial information* for P and *total information* for Q [R]. Since the FSPs are acyclic, the game is a finite one.

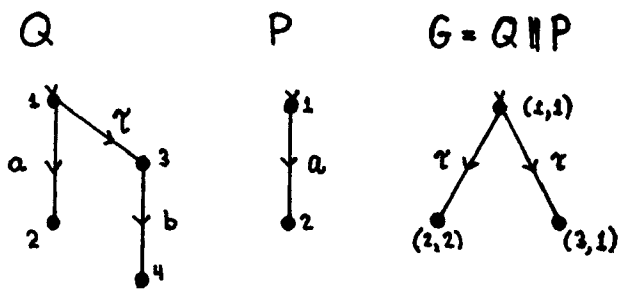


Figure 3

Game(P, Q)

Initial Position: Network of FSPs $\{P, Q\}$, Σ is common alphabet.

$p_1 \leftarrow start\_state\_of\ P$
$q_1 \leftarrow start\_state\_of\ Q$.

Round i $(i = 1, 2, \cdots, n)$:

Player-Q: Sets $a_i \leftarrow a$, $q_{i+1} \leftarrow q$, where $a \in \Sigma$, $q_i \stackrel{a}{\Rightarrow} q$, and $\exists p' \,.\, p_i \stackrel{a}{\Rightarrow} p'$.

Player-P: Sets $p_{i+1} \leftarrow p$, where $p_i \stackrel{a_i}{\Rightarrow} p$.

Information: Q knows everything. P knows initial position and sequence of a_i's.

Goal: Player-P wins iff p_i is *leaf_of* P.

Assumptions: Players have to play if they can.
The FSP P has no τ-moves.

Figure 4

3.2. Complexity

In this section we will examine the complexity of deciding the predicates $S_u(P, Q)$, $S_a(P, Q)$, and $S_c(P, Q)$, where we are given a network $N = \{P_1, P_2, \cdots, P_m\}$ of acyclic FSPs; $P = P_1$ and $Q = P_2 \,||\, P_3 \,||\, \cdots \,||\, P_m$. N is of size n.

For arbitrarily connected networks of linear FSPs there are no significant choices to be made and we can make the following observation:

Proposition 1: If all processes in N are linear, then:

$S_u(P, Q) = S_a(P, Q) = S_c(P\ Q)$ and can be decided in $O(n)$ time.

Proof: To see this, first construct a directed graph H consisting of m linear orders. The nodes of H are all the non-τ transitions of the processes, each node labeled by its action symbol. If transition δ is the first non-τ transition of P_i before the non-τ transition δ' of P_i, insert the arc (δ, δ') in H. Now match the nodes of H in pairs. That is, match the first node with label a in P_i with the first node with label a in P_j $(a \in \Sigma_i \cap \Sigma_j)$, etc. Delete all unmatched nodes and their successors from H until the only remaining nodes are all matched in pairs (the deleted actions have no hope of ever being executed). If a node of P_1 has been deleted, we can now say $S_c(P, Q) = false$. Otherwise, create the directed graph H' by merging each matched pair into one node and keeping only the matched pairs that are predecessors of some matched pair with a node in P_1. H' is acyclic iff $S_c(P, Q) = true$, and H' is cyclic iff $\neg S_u(P, Q) = true$. ∎

For the case of acyclic FSPs both $S_c(P, Q)$ and $\neg S_u(P, Q)$ are in NP. This is because the desired "witness" in each case, which we use to verify success with collaboration and potential blocking respectively, is an $O(m \cdot n)$ sequence of moves from the initial global state (i.e. the length of the maximum path in the global machine G).

For $S_a(P, Q)$, however, we have a finite game of linear time bounded alternation (i.e. the length of the maximum path (i.e. the length of the maximum path in the distinguished machine P). Thus $S_a(P, Q)$ is in PSPACE.

Unfortunately, even simple choices can lead to a combinatorial explosion.

Theorem 1: Let the processes in N be acyclic and such that $|\Sigma_i \cap \Sigma_j| \leq 1$ for $1 \leq i \neq j \leq m$. $S_c(P, Q)$ and $\neg S_u(P, Q)$ are NP-complete even if either

(1) C_N is a tree and all FSPs but one are $O(1)$ linear FSPs, or

(2) each P_i, $1 \leq i \leq m$, is an $O(1)$ tree FSP. ∎

In case (1), the result holds both if P_1 is the only acyclic but non-linear FSP or if P_1 is linear. Note that communication between two processes in Theorem 1 is restricted to repetitions of just one symbol.

The reductions are from 3SAT, where each variable is restricted to appear once negated and once or twice unnegated, a well-known NP-complete problem. They are illustrated in Figure 5 for case (1) and Figure 6 for case (2) using the formula $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$.

Theorem 2: Let the processes in N be acyclic. Even if C_N is a tree and all processes except P are tree FSPs, $S_a(P, Q)$ is PSPACE-complete. |

The reduction is from QBF [GJ] and is illustrated in Figure 7 using the formula $\exists x_1 \forall x_2 \exists x_3 \ (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$.

3.3. Efficient Tests for Tree Processes

The computational complexity indicated by Theorems 1 and 2 depends on two factors. One is high connectivity in C_N (Theorem 1, case 2). The other is the fact that an acyclic process can succinctly describe many sequences of events (Theorem 1, case 1 and Theorem 2). In a loosely connected network, for example whenever C_N is a k-tree for some fixed k, the second factor is critical.

Using the algebraic properties of *composition* and the notion of *possibilities* we can show the following:

Theorem 3: Let N be a network of *tree* processes of size n and let C_N be a k-tree. Then, $S_u(P, Q)$, $S_a(P, Q)$, $S_c(P, Q)$ can be decided in $O(n^k)$ time. |

We first have to express the various properties of interest using the notions of $Lang(\cdot)$ and $Poss(\cdot)$.

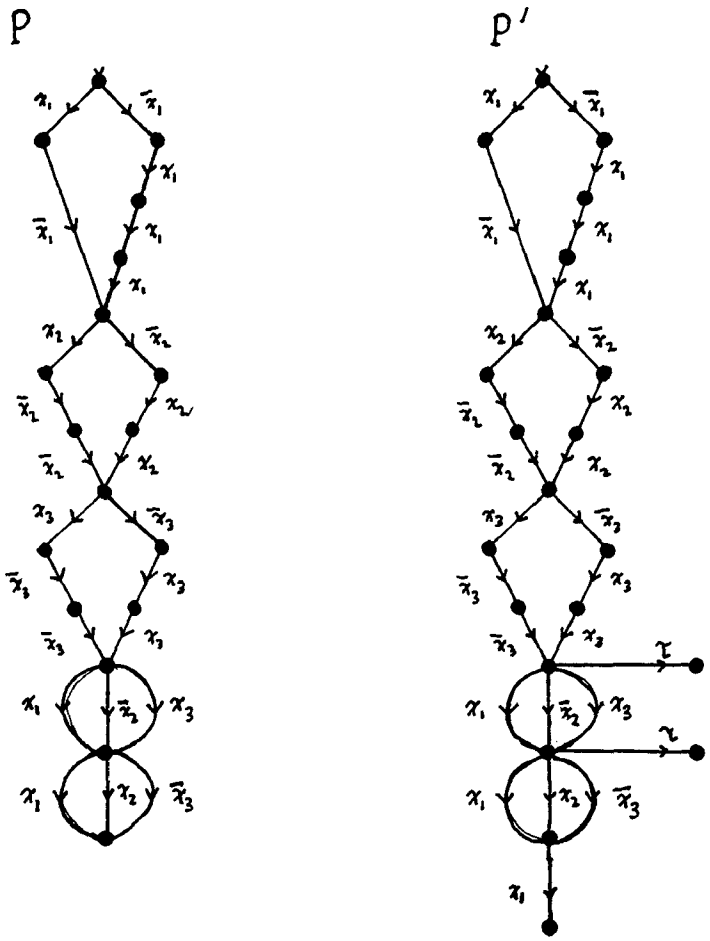


Figure 5

for $S_c(\cdot)$ use $P_1 = P$
for $\neg S_u(\cdot)$ use $P_1 = P'$

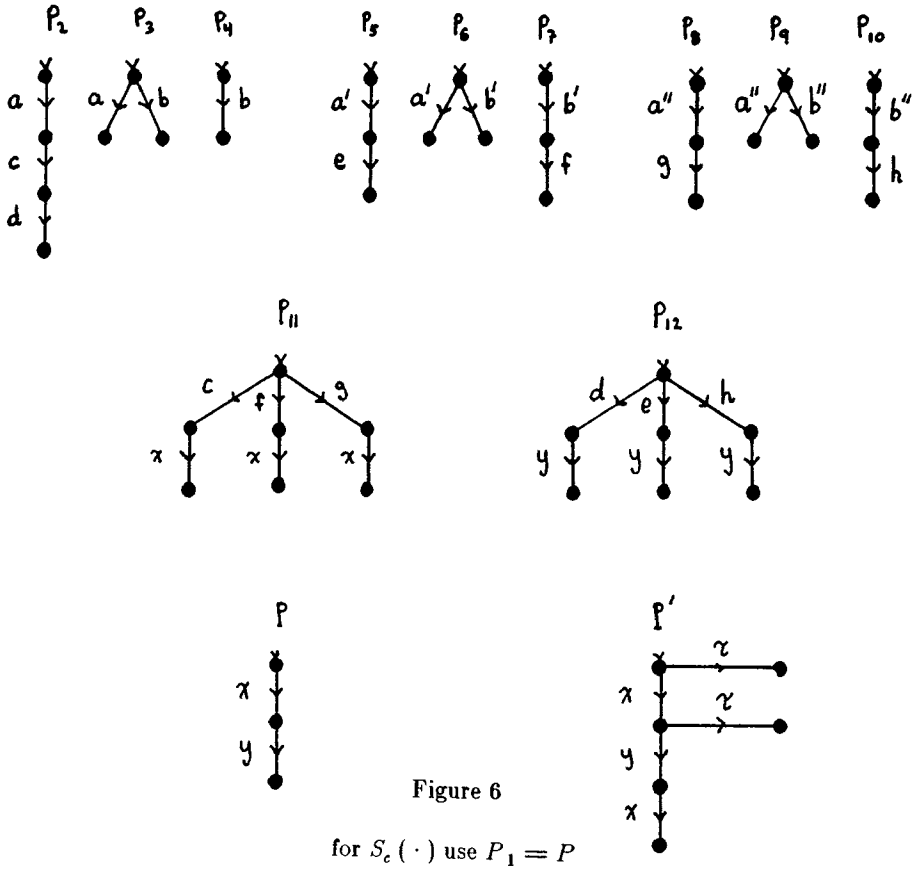


Figure 6

for $S_c(\cdot)$ use $P_1 = P$
for $\neg S_u(\cdot)$ use $P_1 = P'$

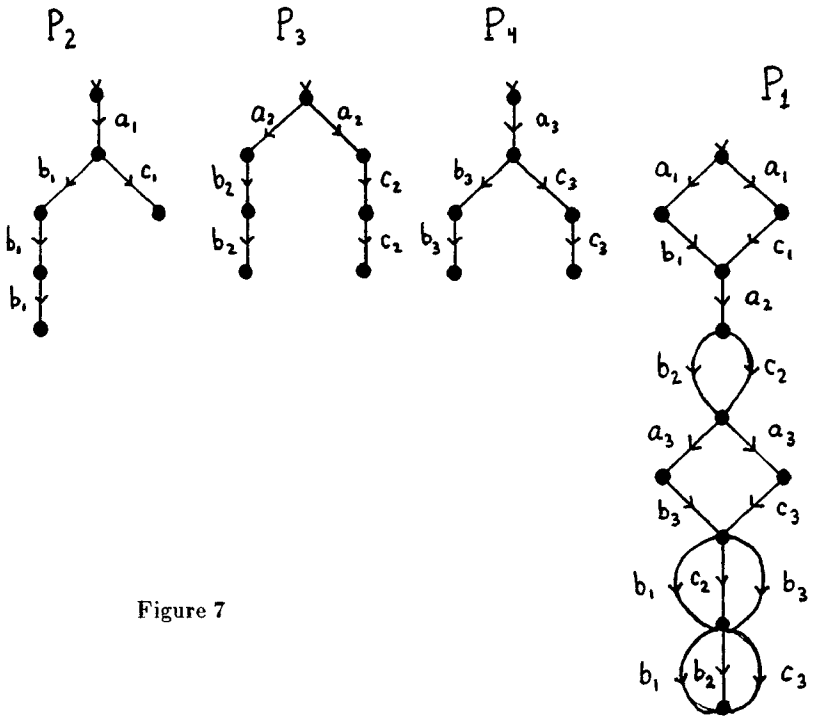


Figure 7

Lemma 3: (success with collaboration) $S_c(P, Q) = true$ iff

$$\exists s.\ \ s \in Lang(Q) \text{ and } (s, \emptyset) \in Poss(P). \quad |$$

Lemma 4: (potential blocking) $\neg S_u(P, Q) = true$ iff

$$\exists s, X, Y.\ \ (s, X) \in Poss(P) \text{ and } (s, Y) \in Poss(Q) \text{ and } X \neq \emptyset \text{ and } X \cap Y = \emptyset. \quad |$$

Both lemmas are simple consequences of the definitions in Section 3.1. The more interesting notion is blocking, which is very much related to possibilities.

Lemma 5: (success in adversity)

$$Poss(Q) = Poss(Q') \text{ implies } S_a(P, Q) = S_a(P, Q'). \quad |$$

As will be made clear by the analysis of an optimal strategy for player P, no distinction can be made between Q and Q'. This is because player P uses only partial information, whereas Q (Q') has total information.

Proof of Lemma 5: Let us assume we are player P at round i, that string $s = a_1 a_2 \cdots a_i$ has been exchanged, and that we are in state p_i. We wish to decide if it is a winning choice to use a_i and go to state p_{i+1} (we call this situation $<s, p_i, p_{i+1}>$). We examine the immediate descendants of p_{i+1} for all of which we have inductively solved the question. That is, we know if situation $<s\sigma, p_{i+1}, p'>$ is a winning one or not, where $\sigma \in \Sigma$ and $p_{i+1} \stackrel{\sigma}{\Rightarrow} p'$. The basis of this induction are the leaves of the FSP P and all paths leading to them; these are winning situations for us. Let the outgoing actions σ to winning situations form set W and to loosing situations form set L.

If there is a possibility $(s, Z) \in Poss(Q)$ such that $Z \cap (W \cup L) = \emptyset$, player Q can block us and the situation is a loosing one. If there is a string $s\sigma \in Lang(Q)$ such that $\sigma \in L - W$, then player Q can force us into a loosing position and thus the situation is again a loosing one. Otherwise, because of the continuity rule, player Q must give us an action for which we have a winning choice.

Finally, it is clear that player P need only examine situations with s in $Lang(Q)$. Recall that $Poss(Q) = Poss(Q')$ implies $Lang(Q) = Lang(Q')$. |

Proof of Theorem 3: We can assume, without loss of generality, that C_N is a tree. This is because the processes in a given partition of a k-tree can be composed directly to produce $O(n^k)$ size processes. Since $||$ is associative and commutative, there is no restriction on how it is applied. (For example, a ring network can be transformed into a path of $O(n^2)$ size processes – see Figure 8a.) We thus modify C_N to obtain $C_{N'}$ so that in the tree $C_{N'}$ the distinguished process P is at the root and the processes, whose product is Q, form a forest of trees: $\{T_1, T_2, \cdots, T_l\}$, where P_{ij} is the jth process in tree T_i and $Q_i = \prod_j P_{ij}$, and $Q = \prod_i Q_i$.

We will now further reduce the tree $C_{N'}$ to a star configuration with P at the center by replacing each Q_i (and tree T_i of FSPs) with a Q_i' (a single FSP), such that $Poss(Q_i) = Poss(Q_i')$. By Lemma 2, $Poss(Q) = Poss(\prod_i Q_i) = Poss(\prod_i Q_i')$. By Lemmas 3, 4, and 5 there is no change in the various success predicates.

We construct Q_i' using the processes in the tree T_i; we proceed from T_i's leaves. Let us start with a node P_f in T_i which communicates with leaves $\{P_c \mid P_c$ is *leaf_of* T_i and *child_of* $P_f\}$, and with one other process in T_i, say P_g (see Figure 8b). If $Poss(P_c) = Poss(P_{cn})$, i.e. P_{cn} is a normal form of P_c with respect to possibilities, we have by Lemma 2 that $Poss((\prod P_c) || P_f) = Poss((\prod P_{cn}) || P_f)$. The critical step now is to compute a normal form P_{fn} such that $Poss(P_{fn}) = Poss((\prod P_{cn}) || P_f)$. For tree processes this is possible with P_{fn} having size no greater than P_f and using linear time in the size of P_f and the P_{cn}'s.

Reduction Step:

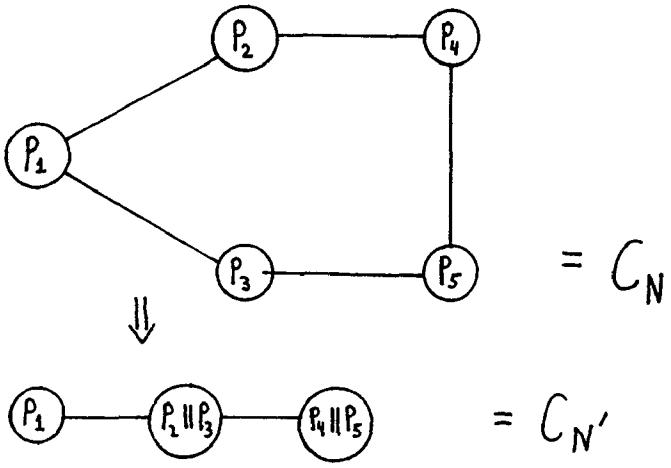
Let $(s, Z) \in Poss((\prod P_{cn}) || P_f)$. Then s is a string over $\Sigma_f \cap \Sigma_g$ and $Z \subseteq \Sigma_f \cap \Sigma_g$ (Σ_f, Σ_g, and Σ_c are the alphabets of P_f, P_g, and P_c). Moreover, there is a "witness" string r to this fact, which is a string in $Lang(P_f)$. The symbols in r from $\Sigma_f \cap \Sigma_g$ form s, the symbols in r from each $\Sigma_f \cap \Sigma_c$ form s_c. The witness string r takes P_f from the start state to some state p' with no outgoing τ-moves in P_f, exactly Z as outgoing actions in $\Sigma_f \cap \Sigma_g$, and outgoing actions W_c from each $\Sigma_f \cap \Sigma c$ which are "appropriate". By appropriate we mean that for each P_c there is a possibility (s_c, X_c) such that $(X_c \cap W_c) = \emptyset$.

It is now clear that if P_f is a tree and the P_{cn}'s are (inductively) of size no greater than the size of P_c, then the possibilities (s, Z) forming P_{fn} can be computed in linear time and will be a set of no greater size than P_f (Figure 9 presents an example).

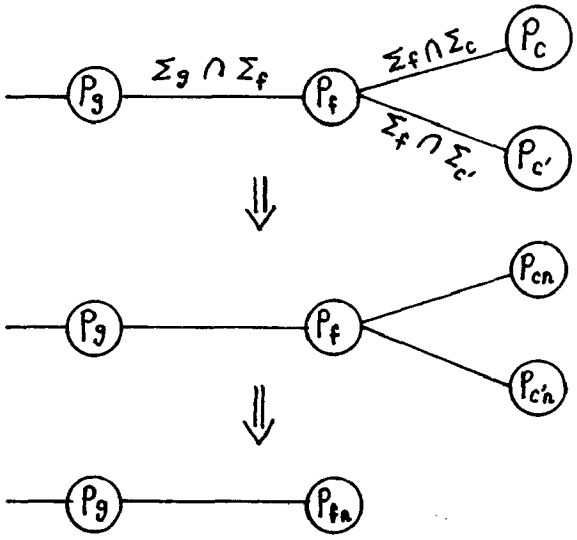
Final Step:

We have reduced the three decision questions of interest to decision questions for a star network of tree processes with P at the center, Q_i''s as leaves, and each tree process of size $O(n^k)$.

Using Lemmas 3, 4 and the proof of Lemma 5, as well as the "small" size of $Poss(P)$ (recall that P is a tree), we can decide $S_u(P, \prod Q_i')$, $S_a(P, \prod Q_i')$, and $S_c(P, \prod Q_i')$ in $O(n^k)$ time. This is because, although $\prod Q_i'$ might be large, there is no interaction between the small Q_i'



(a)



(b)

Figure 8

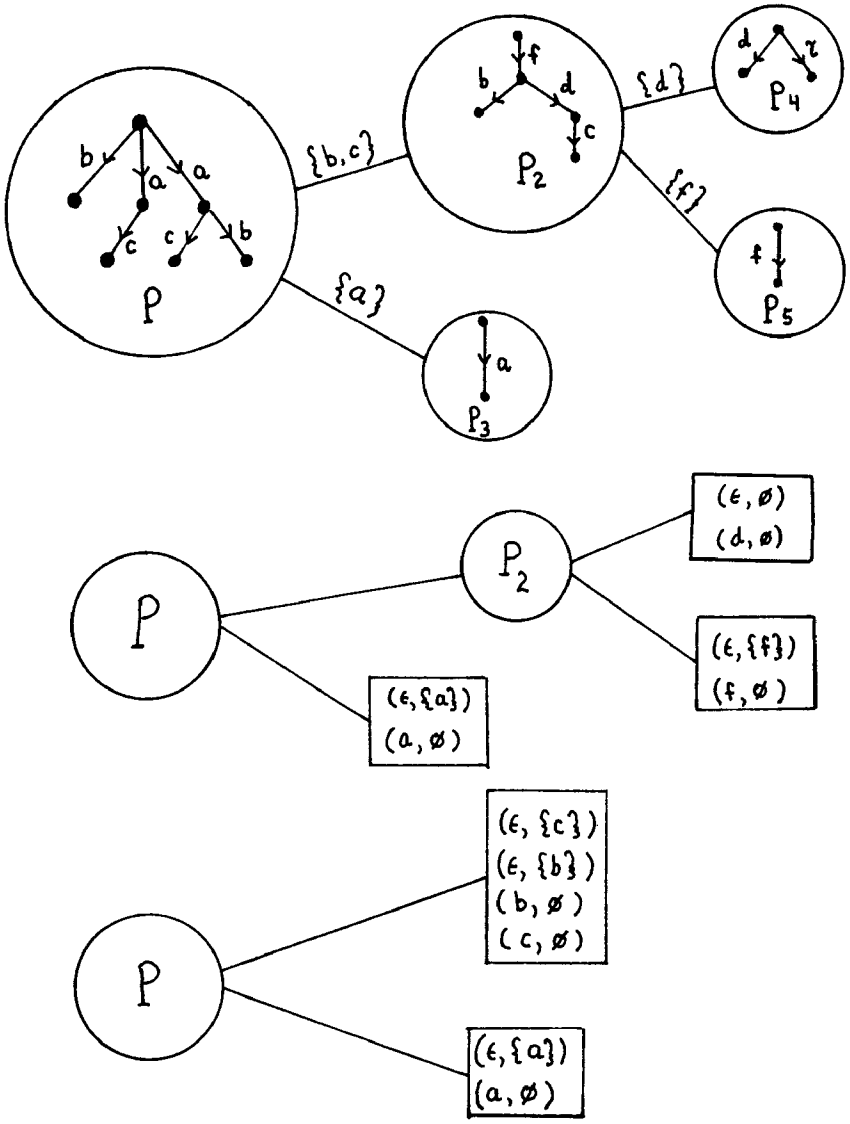


Figure 9

$S_u(P, Q) = false$ (P_4 makes τ-move; P left branches on a)
$S_a(P, Q) = true$ (P right branches on a)
$S_c(P, Q) = true$

4. Cyclic Processes

In this section we will extend most of the theory of Section 3 to cyclic processes. We will assume throughout that $N = \{P_1, P_2, \cdots, P_m\}$ is the given network of processes of size n, $P = P_1$ is the distinguished process, and $Q = P_2 || P_3 || \cdots || P_m$ its context. We will also assume that each P_i in N has no τ-moves and no leaves. These assumptions simplify our arguments, without loss of generality. Note that even if $P_2, \cdots, P_m$ have no leaves and no τ-moves, Q very well could have such features. Once again the continuity rule (Section 3) forces the evolution of the system.

The goals of success of the distinguished process P in context Q become:

(a) P cannot stop moving;

(b) P can keep moving forever even if Q is antagonistic;

(c) P can keep moving forever if Q collaborates with it.

This is a natural generalization of Section 3 if process P is thought of as an infinite tree whose leaves are at infinity.

4.1. Success and Possibilities

The technical problem of the generalization from finite to infinite trees is the presence inside a process, say process Q, of a cycle of τ-moves. Intuitively, such a cycle means that if Q can reach this cycle, it can make choices that will keep it there forever. The behavior of such a temperamental Q would then be similar to going, via a τ-move, to a new leaf. There is a simple solution to this problem through a modification of the composition operator $||$.

Composition for Cyclic Processes Q_1 and Q_2: As in Section 2.2, we first construct $Q_1 \times Q_2$, then keep only the reachable part $Q_1 \cap Q_2$, and then hide the common symbols in $Q_1 || Q_2$. The addition is that if (q_1', q_2') is a state in $Q_1 || Q_2$ from which, using τ-moves, we can enter a loop of τ-moves, then we add a new leaf q_{new} to the state space and a new τ-move $((q_1', q_2'), \tau, q_{new})$ to the transitions of $Q_1 || Q_2$.

Consider two cyclic processes R_1, R_2 produced from N using the above composition. A first consequence of the new definition is that $Poss(R_1) = Poss(R_2)$ implies $Lang(R_1) = Lang(R_2)$. Thus we have already overcome one difficulty mentioned in Section 2.2.

Let us assume that we use the above composition operator $||$ to produce processes from our network N. Let us then use Definition 4 for $Lang(\cdot)$ and $Poss(\cdot)$ of these processes. Note that the new $||$ is still associative and commutative.

Lemma 2′ :

$Poss(R_1) = Poss(R_2)$ implies
$Poss(R || R_1) = Poss(R || R_2)$, and
$Lang(R_1) = Lang(R_2)$ implies
$Lang(R || R_1) = Lang(R || R_2)$.

Sketch of Proof: The proof is similar to that of Lemma 2 except for one case. This is for $(s, \emptyset) \in Poss(R || R_1)$ that was produced because of a *new* τ-loop reachable using s in $R || R_1$. This possibility $(s, \emptyset)$ is present because the new $||$ operator forces us to make a τ-move to a new leaf in $R || R_1$.

Let $s = \epsilon$ (without loss of generality). Then this particular possibility $(\epsilon, \emptyset)$ implies that there is an *infinite* set of strings in $Lang(R) \cap Lang(R_1)$, and also in $Lang(R) \cap Lang(R_2)$. Using this infinite set, the finite-state R_2, and the pigeon-hole principle, we can argue that $(\epsilon, \emptyset) \in Poss(R || R_2)$. |

We can now define reasonable generalizations of the success predicates.

$\neg Su(P, Q) = true$ **(potential blocking)**
iff
$\exists s, X, Y.\ (s, X) \in Poss(P)$ and
$(s, Y) \in Poss(Q)$ and $X \cap Y = \emptyset$.

$Sc(P, Q) = true$ **(success with collaboration)**
iff
$Lang(P) \cap Lang(Q)$ is infinite.

$Sa(P, Q) = true$ **(success in adversity)**
iff
P has a winning strategy in $Game(P, Q)$.

Game (P, Q) **for cyclic processes has the same rules as the game in Figure 4. The only difference is that if** *the game stops* Q is the winner. P wins if it can force *the game to go on forever.* Since we have modified composition such that if Q has the option to get in a τ-loop forever, it now has the option to reach a leaf and win, **we have lost no generality.**

As with *Game*, **potential blocking and success in** collaboration are natural generalizations of **Lemmas 3 and** 4. Again the new $||$ handles τ-loops.

4.2. Complexity

A reasonable assumption about a network N of FSPs is that C_N is a k-tree. It is also reasonable to assume that the processes are cyclic processes "small" in comparison to the size of the network. Under these assumptions, the methodology of Theorem 3 could be used, where $Poss(\cdot)$ and $Lang(\cdot)$ supply the normal form notions. Unfortunately, in the worst case, **the** reduction step and final step (see proof of Theorem 3) **are** no longer efficient. This is because testing possibility equivalence of cyclic processes is PSPACE-complete [KS].

Proposition 2: $S_u(P, Q)$ and $S_c(P, Q)$ are PSPACE-complete, even if N consists strictly of constant size cyclic processes and C_N is a tree. $S_a(P, Q)$ can be decided in deterministic time d^n for some $d > 1$, and cannot be solved in deterministic time $c^{n/\log n}$ for some $c > 1$. |

The reductions are from LBA acceptance [GJ] and alternating linear space Turing machine acceptance. We use techniques similar to [RT] and [L]. Binary communication alphabets are needed. These worst case bounds are not surprising, however, using the powerful combinatorial result in [Le] we can show that:

Theorem 4: Let $N = \{P_1, P_2, \cdots, P_m\}$ be a network of $O(1)$ size cyclic processes such that C_N is a tree. Also let $|\Sigma_i \cap \Sigma_j| \leq 1$ for $1 \leq i \neq j \leq m$. Then:

$S_c(P, Q)$ can be decided in $O(m^k)$ time for some constant k.

Sketch of Proof: The method is similar to that of Theorem 3. Since $S_c(P, Q)$ depends only on $Lang(\cdot)$, we need a normal form weaker than the one preserving $Poss(\cdot)$. A normal form preserving $Lang(\cdot)$ in the case of a unary communication alphabet is just a number: if we have a finite prefix-closed language, it is the length of the longest string; else it is everything. Note that this number must be coded in binary, for it is easy to construct a chain of multiply-by-2 processes.

Reduction Step: P_f is now a constant size machine to which we present a constant number of $O(m)$ bit integers. P_{fn} will be an $O(m)$ bit integer (see Theorem 3 and Figure 8b). Because P_f is a finite state machine, we can formulate the computation of P_{fn} as an integer programming problem with a constant number of variables and use the [Le] polynomial time algorithm. (The argument for the final step is similar). |.

5. Discussion and Open Problems

We have demonstrated the use of a new algebraic method for analyzing communicating finite state sequential processes. The basis of the method is the introduction of an appropriate composition operator (e.g. $||$) with the right normal form (e.g. $Poss(\cdot)$). This interaction allows us to use the structure of a system of processes in order to decide nontrivial properties about the system behavior. These properties are both about cooperating and antagonistic processes.

Our approach leads to efficient time bounds for all properties in the finite tree case. We also extend the approach in a natural fashion to the more practical cyclic case. We believe that despite the worst case complexity of the cyclic case, the technique can be useful in practice.

An interesting generalization of these problems involves introducing more symmetry between the distinguished process and the other processes in the network. For this, let $N = \{P_1, P_2, \cdots, P_{2m}\}$ be a network partitioned into $P = P_1 || P_2 || \cdots || P_m$ and $Q = P_{m+1} || \cdots || P_{2m}$. Deciding success for tree processes in this case is an open question.

Acknowledgements: The authors are indebted to Steve Brookes, Ed Clarke, and Ashfaq Munshi for helpful discussions.

References

[Br] S.D. Brookes, "On the Relationship of CCS and CSP", *Proceedings of 10th ICALP*, Barcelona, Spain, pp. 83-96 (July 1983).

[BZ] D. Brand, P. Zafiropoulo, "On Communicating Finite-State Machines", *Journal ACM*, Vol. 30, No. 2, pp. 323-342 (1983).

[CES] E.M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach", *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*, Austin, TX (April 1983).

[GJ] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco (1979).

[H] G.J. Holzmann, "A Theory for Protocol Validation", *IEEE Transactions on Computers*, Vol. C-31, No. 8, pp. 730-738 (Aug. 1982).

[HBR] C.A.R. Hoare, S.D. Brookes, A.W. Roscoe, "A Theory of Communicating Sequential Processes", *Journal ACM*, Vol. 31, No. 3, pp. 560-599 (July 1984).

[KS] P.C. Kanellakis, S.A. Smolka, "CCS Expressions, Finite State Processes, and Three Problems of Equivalence", *Proceedings of 2nd ACM Symposium on the Principles of Distributed Computing*, Montreal, Canada, pp. 228-240 (Aug. 1983).

[L] R. Ladner, "The Complexity of Problems in Systems of Communicating Sequential Processes", *Journal of Comput. Systems Science 21*, No. 2, pp. 179-194 (1980).

[Le] H.W. Lenstra, Jr., "Integer Programming with a Fixed Number of Variables", Mathematics Department Report 81-03, University of Amsterdam (1981).

[M] R. Milner, "A Calculus of Communicating Systems", *Lecture Notes in Computer Science 92*, Springer-Verlag (1980).

[OL] S.S. Owicki, L. Lamport, "Proving Liveness Properties of Concurrent Programs", *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 455-495 (July 1982).

[R] J.H. Reif, "Universal Games of Incomplete Information", *Proceedings of 11th ACM Symp. on Theory of Computing*, pp. 288-308 (1979).

[RT] T. Raeuchle, S. Toueg, "Exposure to Deadlock for Communicating Processes is Hard to Detect", Department of Computer Science, Technical Report No. TR 83-555, Cornell University, Ithaca, NY (May 1983).

[S] S.A. Smolka, "Analysis of Communicating Finite-State Processes", Department of Computer Science, TR No. CS-84-05, Brown University, Providence, RI (Feb. 1984).

[T] R.N. Taylor, "Complexity of Analyzing the Synchronization Structure of Concurrent Programs", *Acta Informatica 19*, pp. 57-84 (1984).