The Complexity of Reachability in Distributed Communicating Processes*

John H. Reif[1,**] and Scott A. Smolka[2,***]

[1] Department of Computer Science, Duke University, Durham, NC 27706, USA
[2] Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400, USA

Summary. A crucial problem in the analysis of communicating processes is the detection of program statements that are unreachable due to communication deadlocks. In this paper, we consider the computational complexity of the reachability problem for various models of communicating processes. We obtain these models by making simplifying assumptions about the behavior of message queues and program control, with the hope that reachability may become easier to decide. Depending on the assumptions made, we show that reachability is undecidable, requires nearly exponential space infinitely often, or is NP-complete. In obtaining these results, we demonstrate a very close relationship between the decidable models and Petri nets and Habermann's path expressions, respectively.

1. Introduction

1.1 The Problem

We consider a system of distributed concurrent processes, each sequentially executing a distinct program and communicating by the transmission and reception of messages. We assume that there is no interference between processes by shared variables, interrupts, or any other synchronization primitives beyond the message primitives.

Various channels are available for communication among processes, and each channel has a unique process which is the destination of messages transmitted via this channel. Communication among processes is *static* if the channel arguments to message primitives are constants, and otherwise is *dynamic*. Hoare's

* A preliminary version of this paper appeared in the proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages, pp. 257–268, June 1979

** Supported by National Science Foundation Grant NSF MCS82-00269 and the Office of Naval Research Contract N00014-80-C-0647

*** Supported by National Science Foundation Grants NSF DCR-8505873 and CCR-8704309

Preprint of paper appearing in Journal of Acta Informatica, Vol. 25(3), April 1988, pp.333-354.

CSP [9] is a language proposal incorporating static communication, while NIL [25], a high-level systems programming language developed at IBM Research, Yorktown Heights, uses dynamic communication. We consider both static and dynamic communication.

We assume a semantics for message passing where the process transmitting a message need not wait until reception of the message, and thus an unbounded queue of yet-to-be-received messages is associated with each channel. This type of *asynchronous* message passing is used in NIL and in the distributed programming language PLITS [4].

An analysis problem of particular interest here is *reachability*: Can a given program statement ever be reached in some execution? Reachability is perhaps the most fundamental of all analysis problems. If a program statement n of a process P is unreachable, then, as we show, an attempt by process P to reach n will result in a deadlock. Many other analysis problems for communicating processes require detection of reachable statements, such as data flow analysis [21].

In this paper, we consider the computational complexity of the reachability problem for various models of communicating processes. Let M_0 be the basic semantic model in which each process executes a program with conditional tests, and the message passing semantics are as described above. We consider weakened versions of M_0 that allow all executions valid in M_0, but may also allow additional, spurious executions. A statement that is unreachable in one of the weakened models is, of course, also unreachable in the more powerful semantics of M_0. Thus, a reachability analysis performed in a weakened model may be termed *conservative*.

We are not necessarily interested in models strong enough for correctness proofs. Instead we desire *reasonable models* for which analysis problems are decidable, and for which algorithms exist that are powerful enough to be useful for practical situations of program analysis.

1.2 Summary of Results

Obviously, reachability in M_0 is undecidable. Even if we allow a single process to execute conditionals, reachability of statements is undecidable (we can directly simulate a universal machine). With M_0 as a point of reference, what is the complexity of the reachability problem if we weaken the semantics by disallowing conditionals?

In Sect. 2 we describe our *flow graph model* M_1 for systems of communicating processes. Here the flow of control of a process's program is represented by a *program flow graph* as is usual in data flow analysis [e.g., 8]. This model uses branching in the flow graph where conditional tests existed in the original program. Consequently, the set of executions modeled by the flow graph are a superset of the executions of the underlying program. The message queues of M_1 are FIFO and messages are deleted from message queues by receive statements. Unfortunately, we show in Sect. 3 that testing reachability in M_1 is recursively undecidable. However, in the case of static communication, testing reachability in M_1 is shown to be polynomial time reducible to and from Petri

net coverability. This problem is known to be decidable in deterministic exponential space [19] but requires nearly exponential space, infinitely often [13, 14, 23].

In Sect. 2 another model M_2 is introduced with simplified communication semantics: receive statements *copy* rather than delete messages from message queues. In Sect. 4 we characterize executions in M_2 with static communication by extended regular expressions similar to the path expressions of Habermann [7]. For each program statement, there is an extended regular expression that generates the empty language iff the given program statement is unreachable in M_2. We then show that reachability in this model is NP-complete with a reduction from satisfiability of boolean formulas in three-conjunctive normal form.

1.3 Related Work

Since the results in this paper were originally published, the decidability and complexity of reachability-related problems for various models of communicating processes have been examined. One very popular model is networks of Communicating Finite State Machines (CFSM's) where message channels are unidirectional, unbounded, and FIFO. Conditional control flow constructs are not present in this model, but transmit and receive statements are conditional in the sense that they specify a particular type of message to be communicated. Numerous problems have been analyzed in the context of CFSM's including reachability, boundedness, deadlock, livelock, unspecified receptions, nonexecutable receptions, and stable states. An early result by Brand and Zafiropulo [1] showed that, in general, the boundedness problem for CFSM's is undecidable; a Turning machine simulation was used. This undecidability result also holds for non-FIFO channels [20]. Cunha and Maibum [3] later demonstrated that the problem becomes decidable if only one type of message is allowed. Yu and Gouda [29] improved upon this result by exhibiting a more efficient algorithm. Pachl [17] also proved decidability, this time for message channels whose behavior can be described by "rationale expressions".

Gouda et al. [5] considered the deadlock detection problem for networks of two CFSM's. They showed that the problem is PSPACE-complete if one of the channels is bounded by a linear function of the input size, and NLOG-SPACE-complete if one of the channels is bounded by some fixed constant. In [20], Rauchle and Toueg proved PSPACE-hardness for the case where both channels are bounded, using a reduction from the word problem for context sensitive languages.

Gouda and Rosier [6] studied priority networks of CFSM's where messages are received according to a fixed, partial-order priority relation (unrelated messages can be received in any order). They showed, using a simulation of a 2-counter machine without input, that the problems of detecting deadlock and boundedness in priority nets with two or less message types are undecidable. They also proved that if the priority relation is the null set (i.e., all messages are received on a random basis), then the boundedness problem is decidable by reduction to the boundedness problem for vector addition systems.

Rosier and Yen [24] considered networks of CFSM's that explicitly allow zero testing (i.e., empty channel detection) and showed that the boundedness problem is decidable if only a single type of message is communicated.

Results have also been obtained for models in which finite-state processes must synchronize in order to communicate, i.e., there are no message queues. Ladner [12] showed that in this context the "lockout problem", which can be viewed as a two-player game, requires exponential time to solve. In [28], Taylor proved that the "possible rendezvous" problem is NP-complete for acyclic processes. The work of Kanellakis and Smolka [11] refined the results of [28] in a slightly different setting.

A companion paper [21] describes an iterative technique for data flow analysis of communicating processes, which generalizes previously known techniques for data flow analysis of sequential programs. This analysis can be used to determine useful properties of communicating processes, such as bounds on the values of variables and messages.

1.4 Computational Complexity Definitions

We shall view a computational problem as a language recognition problem. Let Σ be a finite alphabet with at least two members. For languages $L, L' \subseteq \Sigma^*$, a *reduction* from L to L' is a function $f: \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$,

$$x \in L \quad \text{iff } f(x) \in L'.$$

L is *recursively reducible* to L' if there exists a computable reduction from L to L', and L is *polynomial time reducible* to L' if there exists a reduction from L to L' computable by a deterministic Turing machine in time polynomial in the length of x, for all $x \in L$.

Let $\mathbf{L}$ be a family of languages from Σ^*. L is $\mathbf{L}$-*hard* if L' is polynomial time reducible to L for all $L' \in \mathbf{L}$, and is $\mathbf{L}$-*complete* if L is $\mathbf{L}$-hard and $L \in \mathbf{L}$.

We let SPACE($S(n)$) denote the family of languages recognizable in deterministic space $S(n)$.

2. Flow Graph Models of Communicating Processes

In this section we present two flow graph models of distributed communicating processes, M_1 and M_2, and their operational semantics. We define the reachability problem relative to these semantics. Model M_2 is obtained from M_1 by simplifying the behavior of message queues. The complexity of reachability in these models is studied in Sects. 3 and 4.

2.1 Flow Graph Model M_1

We describe here a model for a system of processes $\{P_1, \ldots, P_r\}$ which intercommunicate over channels having names taken from a fixed set C. Each process P_i sequentially executes a distinct program consisting of statements of the form:

(1) *Assignment statements* "$X \leftarrow E$" where X is a program variable local to P_i and E is an expression. This statement has the usual effect of setting X to the result of evaluating E.

(2) *Transmit statements* "**TRANSMIT**(E_1, E_2)" where expression E_1 must evaluate to a message channel $c \in C$, and E_2 evaluates to the message to be transmitted, say M. The message M cannot be a pointer value, but is otherwise unrestricted. In particular, M can be a communication channel (in the case of dynamic communication). E_2 may be absent, in which case some fixed default message is sent. The transmit statement is assumed to be executed without delay, regardless of the number of messages previously transmitted over channel c.

(3) *Receive statements* "$X \leftarrow$ **RECEIVE**(E)" where E must evaluate to a communication channel $c \in C$, and X is an optional program variable local to P_i assigned the value of the message received. If no message is in the message queue for channel c, then the receive statement's execution is blocked until a message is transmitted over channel c.

(4) *No-op* (empty) statements will also be allowed. (In our flow graph models, they will be depicted as empty ovals.)

The sets of program variables local to distinct processes are disjoint, and are assumed to have no shared values. Thus, there is no interference between processes except that induced from our message primitives.

Intuitively, the operational semantics for communicating processes is specified by designating for each channel a message queue $Q(c)$, listing the messages transmitted but not yet received over channel c. A receive statement deletes the current message on the front of the appropriate queue. The order of messages appearing in the queue need only be consistent with the assumption that "sucessive" transmissions over a given channel are received in order of transmission.

The communication primitives **TRANSMIT** and **RECEIVE** defined here are essentially the same as the *asynchronous* message passing used in NIL [25] and PLITS [4]. This is in contrast to the *synchronous* message passing of CSP [9] and occam [10], where the transmitter is required to wait until acknowledgement (by "handshake") of reception of a message; message queues are unnecessary in this semantics. This handshake communication can be synchronized in real-time by the algorithms of Reif and Spirakis [22].

The program executed by process P_i is represented by a flow graph $G_i = \langle N_i, E_i, s_i \rangle$. Each node $n \in N_i$ corresponds to a single (non-control) program statement. The edge set $E_i \subseteq N_i \times N_i$ consists of pairs of nodes between which control may transfer. Thus the key difference between this model and M_0 is that conditional statements are not found in N_i since the control flow is specified by the edges of flow graph G_i. An *execution path* of G_i is a path of G_i beginning at the start node s_i (see Fig. 1 for an example). Occasionally, we will distinguish a final or *exit node* $n_f \in N_i$ from which control may be considered to exit. See Hecht [8] for a description of sequential program flow graph models and their application to program optimization.

We now give a formal operational semantics for our flow graph model of communicating processes. Let $\{P_1, \ldots, P_r\}$ be our system of processes, and let G_i be the flow graph of the program executed by P_i, $1 \leqq i \leqq r$. Assume the processes transmit messages from a given value domain V and that Var_i is the

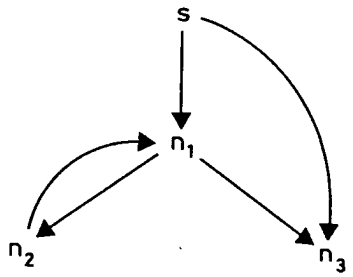


Fig. 1. The above flow graph has execution path (s, n_1, n_2, n_1, n_3), among others

set of program variables local to P_i. We can formally describe the state of the system at any point in its execution in terms of a *global state*

$$S = \langle [n_1, \ldots, n_r], [m_1, \ldots, m_r], \{Q(c) | c \in C\} \rangle.$$

Here:

$n_i \in N_i$ is process P_i's current *locus of control.*

m_i: $\text{Var}_i \to V \cup \{unbound\}$ is P_i's *memory function,* which performs the usual mapping of identifiers to values. We extend m_i to expressions by letting $m_i(E)$ denote the value of expression E based on the bindings in m_i.

$Q(c) \in V^*$ is the current *queue contents* associated with channel $c \in C$, i.e., the sequence of yet-to-be-received messages transmitted over channel c.

The concurrent execution of a system of communicating processes proceeds as the "evolution" of one system global state into another. Such evolution involves the execution of *exactly one* program statement labeling a flow graph node. The execution of a statement is assumed to happen instantaneously and is referred to as an *event.* We use $e_1, e_2, \ldots$ to denote events. System evolution is nondeterministic in that any "enabled" statement may be executed next. Assignment, transmit, and no-op statements are always enabled, i.e., in any global state. A receive statement is enabled in global state S only if $Q(c) \neq \varepsilon$ (the empty channel), where the channel argument to the receive statement evaluates to c. Execution based on system evolution gives us usual nondeterministic interleaving semantics for communicating processes (see [15] for example).

Notation. Let f: $D_1 \to D_2$ be a function. Then, $f[d_2/d_1]$, $d_1 \in D_1$ and $d_2 \in D_2$, denotes the function that is everywhere the same as f, except possibly on d_1 where its value is d_2.

Definition 1. Let $S = \langle [n_1, \ldots, n_r], M = [m_1, \ldots, m_r], B = \{Q(c) | c \in C\} \rangle$ be a global state. Then S *can evolve through event* $e = n_i$ into global state $S' = \langle [n_1, \ldots, n_{i-1}, n_i', n_{i+1}, \ldots, n_r], M', B' \rangle$ iff (n_i, n_i') is an edge in flow graph G_i, and

if n is "$X \leftarrow E$" then

M' equals M with m_i replaced by $m_i[m_i(E)/X]$, i.e., X is now bound to the value of expression E, and B' equals B.

if n is "**TRANSMIT** (E_1, E_2)" then

M' equals M and, assuming E_1 evaluates to $c \in C$, B' equals B with $Q(c)$ replaced by $append(m_i(E_2), Q(c))$; i.e., the value of expression E_2 is appended to the rear of queue $Q(c)$.

if n is "$X \leftarrow$ **RECEIVE** (E)" then
assuming E evaluates to c, M' equals M with m_i replaced by $m_i[head(Q(c))/X]$; i.e., X is now bound to the value at the head of queue $Q(c)$. (If optional program variable X is not present, then no change of variable bindings occur, i.e., $M'=M$.) Also, B' equals B with $Q(c)$ replaced by $rest(Q(c))$, i.e., the head element of $Q(c)$ is removed. Furthermore, for this statement to have been executed in the first place, we must have had $Q(c)\neq\varepsilon$.

if n is a *no-op* then simply M' equals M and B' equals B. □

We also define the *initial global state*

$$S_{\text{init}}=\langle[s_1, \ldots, s_r], [m_1 \Leftarrow unbound, \ldots, m_r \Leftarrow unbound], \{Q(c)=\varepsilon \mid c\in C\}\rangle.$$

Regarding S_{init}, recall s_i is the start node of flow graph G_i; *unbound* is the everywhere *unbound* function (thus, all of memory is initially undefined); and ε represents the empty sequence (thus, all message queues are initially empty). The behavior of the system can then be viewed as an *execution tree* – a possibly infinite, directed, labeled tree of global states rooted at S_{init} such that (S, S') is an edge labeled by e iff S can evolve into S' through event e.

We say that a program statement n is *executable* iff n appears as an edge-labeling event in the system's execution tree. We say that n is *reachable* if it is the start node of a program flow graph, or there exists a flow graph edge (m, n) such that m is executable. Otherwise n is *unreachable*. Clearly the executability of n implies its reachability, but not vice versa: a program statement is reachable as long as flow of control can reach it; the statement itself need not occur as an event.

Finally we define an *execution* of a system of communicating processes to be the sequence of events labeling any finite path in the system's execution tree. We will make use of this definition in Sect. 4, where we characterize the behavior of model M_2 in language-theoretic terms.

Alternatively one can give a partial-order semantics to communicating processes. Let E be the set of possible events. Then $(E, \rightsquigarrow)$ is a partial order of events such that

(1) Events associated with process P_i form a sequential execution of P_i.
(2) If e_{rec} is an event resulting from the execution of statement "**RECEIVE** (E)", and E evaluates to channel c, then e_{rec} must be preceded by a unique event e_{trans} resulting from the execution of a transmit statement whose first argument evaluates to channel c and whose second argument evaluates to the message received; i.e., $e_{\text{trans}} \rightsquigarrow e_{\text{rec}}$. In addition, we have that if e_1, e_2 are events resulting from the reception of messages M_1, M_2, and e'_1, e'_2 are the corresponding transmit events, then $e_1 \rightsquigarrow e_2$ implies $e'_1 \rightsquigarrow e'_2$.

The resulting semantics are nondeterministic in the sense that two simultaneous message transmissions by two processes over the same channel must arrive in sequential order, but we make no assumptions about this order.

In Sect. 3 we show that the question of reachability in model M_1 with dynamic communication is undecidable; for static communication, we show that reachability is polynomial-time reducible to and from coverability of Petri net markings.

2.2 Flow Graph Model M_2 with Simplified Message Queueing Behavior

Here we restrict model M_1 by simplifying the behavior of message queues. In particular, we assume that receive statements *copy* rather than *delete* messages from the message queues. That is, the effect of a receive is to read the message at the head of the specified queue, leaving the contents of the queue intact.

Let M_2 be the model derived from the flow graph model M_1 with this assumption. M_2 seems to exhibit the simplest possible semantics for message queues that is nontrivial. As a consequence once a message channel has been sent a message, any arbitrary number of further receive statements can be executed. However, we do detect in M_2 cases of unreachability that arise when some channel has never been sent a message. Figure 2 demonstrates that the above assumptions result in a model strictly weaker than the flow graph model M_1. The exit node of the depicted flow graph is reachable in model M_2 but not in M_1.

Message queues in model M_2 can be implemented using ordered queues with an append operation only, and a pointer *next_msg* into the queue indicating the next message to be copied. Initially, *next_msg* is one. Regarding Definition 1, only the semantics of receive statements changes. Consider the statement "$X \leftarrow$ **RECEIVE**(E)", and assume E evaluates to c. This statement can be executed only if c is nonempty. If so, the value in position *next_msg* of $Q(c)$ will be copied into X. Then *next_msg* will be incremented by one provided that further messages remain in the queue. Otherwise the same message will be copied by subsequent receive statements until a new message arrives.

In Sect. 4, we show that reachability in model M_2 is decidable but probably not efficiently testable. In fact, we show that reachability in this model with static communication is NP-complete. We do not consider the case of dynamic communication in M_2.

3. Complexity of Reachability in M_1

We consider here the problem of testing reachability in the flow graph model M_1 (Sect. 2.1). In the case of dynamic communication, we show that reachability is undecidable.

A *two-counter machine without input* is an automaton containing a pair of counters which may be incremented, decremented, and tested for zero. The halting problem for two-counter machnes is known to be recursively undecidable [16]. The proof that reachability in M_1 is undecidable uses a recursive reduction from the halting problem for two-counter machines to reachability. This proof exploits the ability of a simple transmit statement to communicate over various channels depending on the evaluation of its channel argument.

Theorem 1. *Reachability in the flow graph model M_1 with dynamic communication is undecidable.*

Proof. Consider a two-counter machine $(S, q_0, q_f, \mathbf{I}, C_1, C_2)$ with state set S, initial state $q_0 \in S$, final state $q_f \in S$, set of instructions $\mathbf{I}$, and counters C_1, C_2. Associated with each state $q \in S - \{q_f\}$ is an instruction $I_q \in \mathbf{I}$ of one of the following forms:

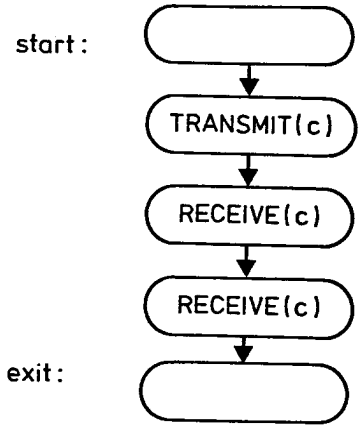


Fig. 2. Exit node reachable in model M_2 but not in M_1

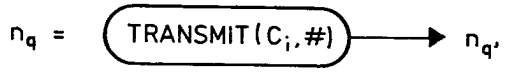
(i) (increment, i, q') where in state q the counter C_i is incremented by 1 and state q' is entered.
(ii) (decrement, i, q') where in state q the counter C_i is decremented by 1 and state q' is entered.
(iii) (test, i, q_1, q_2) where in state q if counter $C_i=0$ then state q_1 is entered and otherwise state q_2 is entered.

A computation begins in the initial state q_0 with both counters set to zero. Computations resulting in transitions to undefined states or negative counters are undefined. The computation *halts* at state q_f.

To simulate this two-counter machine, we build a process P which communicates to and from itself on channels $\{C_1, C_2, \#, \$\}$; so P is the origin and destination of all messages. The flow graph of P is $G=\{N, E, n_{q_0}\}$ and is defined in what follows.

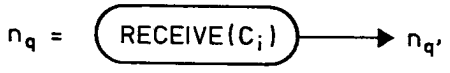
Initially, the message queues $Q(C_1)$, $Q(C_2)$, $Q(\#)$, and $Q(\$)$ are empty. Associated with each instruction $I_q \in \mathbf{I}$ is a subgraph G_q of G which simulates this instruction. We shall claim that if C_i contains the integer $k \geqq 0$, then $Q(C_i) = \#^k$.

(1) If I_q=(increment, i, q') then G_q is of the form:



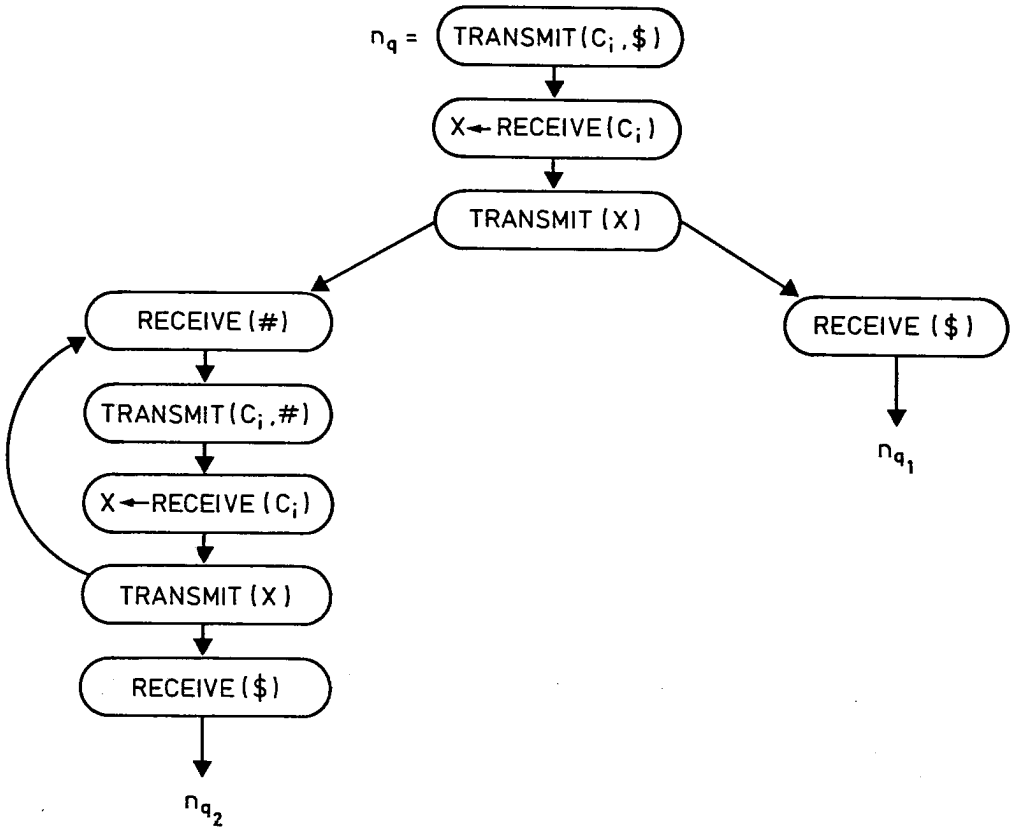
Hence an execution of I_q results in the addition of a # to $Q(C_i)$.

(2) If I_q=(decrement, i, q') then G_q is of the form:



and so an execution of I_q results in the deletion of a # from $Q(C_i)$ if this message queue is not empty. Otherwise, if $Q(C_i)$ is empty, then the process P hangs and is unable to proceed.

(3) If I_q=(test, i, q_1, q_2) then G_q is of the form:



It may be easily shown that if $Q(C_i)$ is empty on execution of q, then n_{q_1} is reachable and otherwise n_{q_2} is reachable. In either case, all queues are restored to their state just before the execution of n_q. Note that the execution of this flow graph depends on the value of X.

This completes the simulation and we thus have that the node n_{q_f} is reachable in some execution of P iff the given two-counter machine halts. □

Next, let us assume that the channel arguments to all transmit and receive statements are constants, i.e., communication is static. With this restriction there is a strong resemblance to a synchronization structure called a *Petri net*. In fact, we show that reachability of statements in this case is polynomial-time reducible to and from coverability of Petri net markings.

A Petri net is a bipartite directed graph $PN=(\pi \cup T, E_{PN})$ with a set of *places* π, a set of *transitions* T, and a set of edges E_{PN} (in general, E_{PN} may be a multiset [18]). An edge is of the form (t, x) or (x, t), where $t \in T$ is a transition and $x \in \pi$ is a place. In the former case, x is said to be an *output place* of t, while in the latter case, x is said to be an *input place* of t. A *marking* of PN is a mapping μ from the places π to the nonnegative integers. Given a marking μ and a transition $t \in T$ with no input places marked by μ with zero, t is *fired* by decrementing the markings of the input places of t by one and incrementing the markings of the output places of t by one. The resulting marking μ' is said to be *reached* from μ. A marking μ is *reachable* from initial marking μ_0 if there exists a sequence of markings $(\mu_0, \mu_1, \ldots, \mu_k=\mu)$ such that μ_i is reached from μ_{i-1} for $i=1, \ldots, k$. A marking μ is *coverable* if there exists a reachable marking μ' such that $\mu'(x) \geq \mu(x)$ for all places $x \in \pi$. A comprehensive exposition on Petri nets can be found in [18].

Processes in model M_1 do not contain conditional statements. However, in the case of dynamic communication, flow of control can still be affected by the order in which messages are enqueued. This situation is evidenced by construction (3) of the proof of Theorem 1. By limiting M_1 to static communication, this dependency on the ordering of messages in message queues disappears: the reachability of a statement in M_1 is not influenced by the order in which messages are enqueued, but only by whether they have been sent or not. Let $\bar{M}_1$ be flow graph model M_1 with the assumption that the message queues are unordered. Consider any two messages m_1, m_2 occurring in this order within a particular queue Q in an execution of model M_1. If we order m_2 *before* m_1 in Q and assume static communication, the resulting execution in M_1 still passes through the same sequence of statements. Thus, we have:

Proposition 2. *In the case of static communication, each statement n is reachable in flow graph model M_1 iff it is reachable in $\bar{M}_1$.* □

Theorem 2. *Reachability in M_1 with static communication is polynomial time reducible to coverability of Petri nets.*

Proof. Let $\{P_1, \ldots, P_r\}$ be a given system of communicating processes with flow graphs $\{G_1, \ldots, G_r\}$. We build a Petri net $PN=(\pi \cup T, E_{PN})$ that simulates this system of processes as follows:

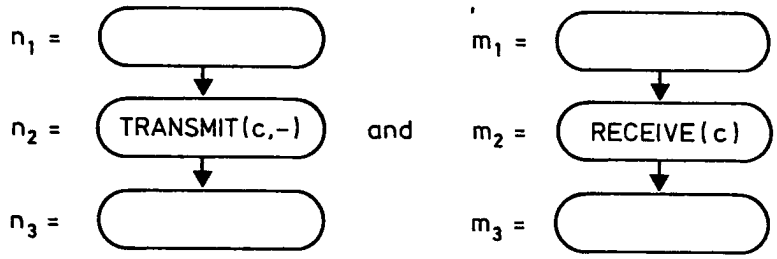
(1) For each channel c occurring as an argument of a transmit or receive statement, we associate a place $Q(c)\in\pi$ called the *message queue counter* of c.
(2) Each node n of each program flow graph G_i is considered a place $n\in\pi$. If n is a statement of the form "**TRANSMIT**(c, E)", then there is also a distinct place $n'\in\pi$, a transition $\bar{n}\in T$, and edges $(n, \bar{n})$, $(\bar{n}, Q(c))$, $(\bar{n}, n')\in E_{PN}$. If n is a statement of the form "$X \leftarrow$ **RECEIVE**(c)", then there is a distinct place $n'\in\pi$ a transition $\bar{n}\in T$, and edges $(n, \bar{n})$, $(Q(c), \bar{n})$, $(\bar{n}, n')\in E_{PN}$.
(3) Each edge (n, m) of each program flow graph G_i, is considered a transition of T and $((n, m), m)\in E_{PN}$. Furthermore, $(n', (n, m))\in E_{PN}$ if n is a transmit or receive statement, and $(n, (n, m))\in E_{PN}$ otherwise.
(4) There are no other places, transitions or edges in PN.

Let n be a node of a program flow graph G_i, and let μ_0 be the marking such that $\mu_0(s_i)=1$, s_i the start node of flow graph G_i, and $\mu_0(x)=0$ for all other places $x\in\pi-\{s_i \mid 1\leqq i\leqq r\}$. It can be easily verified that n is reachable in M_1 with static communication iff there is a marking μ reachable from μ_0 with $\mu(n)\geqq 1$, and $\mu(x)\geqq 0$ for all other places $x\in\pi-\{n\}$. The use of Proposition 2 is crucial here since in the Petri net simulation, only a count of the number of messages residing in any particular queue is maintained. The relative ordering of the messages is lost. □

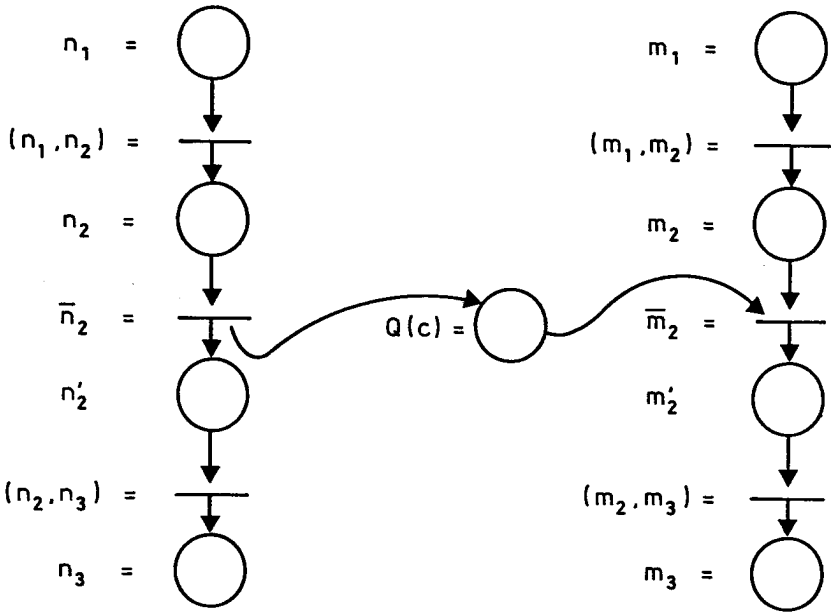
An example of the reduction of Theorem 2 is given in Fig. 3.

Let $n=|\pi|+|T|+|E_{PN}|$ be the input size of a Petri net $PN=(\pi \cup T, E_{PN})$. Rackoff [19] has shown that the coverability problem for Petri nets is decidable in deterministic space $2^{c \cdot n \log n}$, for some constant c. As a consequence, Theorem 2 implies:

Fig. 3. If the following graphs:



are subgraphs of G_1 and G_2, then PN contains as subgraphs:



The initial marking μ_0 has the start node of each G_i marked with one and all other places marked with zero

Corollary 1. *The reachability problem for the flow graph model M_1 with static communication is in* SPACE$(2^{c \cdot n \log n})$. □

Let $PN=(\pi \cup T, E_{PN})$ be a Petri net with initial marking μ_0. We say that PN can be *simulated* in flow graph model M_1 with static communication iff there exists a set of $|\pi|+|T|+1$ program flow graphs $\{G_t | t \in T\} \cup \{G_x | x \in \pi\} \cup \{G_0\}$ such that:

- There is a distinct channel x for each place $x \in \pi$, and channel x initially contains $\mu_0(x)$ messages. All other channels have initially zero messages.
- There exists an execution in M_1 such that G_0 reaches its exit node with exactly $\mu(x)$ messages in each channel $x \in \pi$ iff μ is a reachable marking for PN.

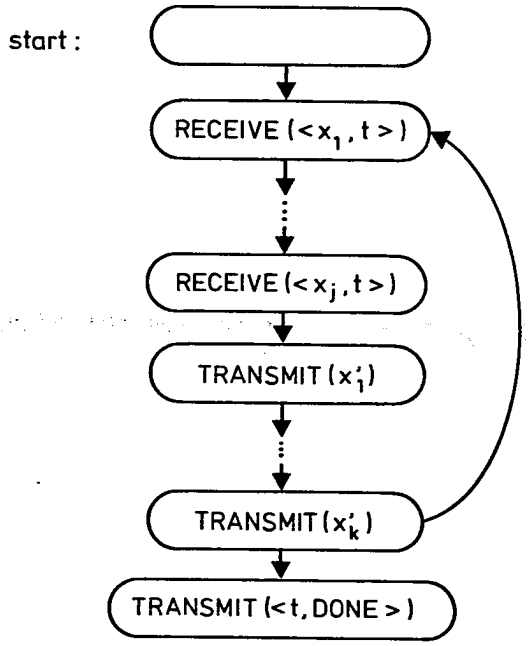
Lemma 1. *Given Petri net PN, we can construct in time polynomial in n a set of program flow graphs that simulate PN.*

Proof. The above-mentioned set of program flow graphs used in the construction will communicate over channels having names from the set

$$\{\langle x_i, t_j\rangle \mid t_j \text{ is a transition of } PN \text{ and } x_i \text{ is an input place of } t_j\}$$
$$\cup \{x_i, \langle x_i, DONE\rangle \mid x_i \text{ is a place of } PN\}$$
$$\cup \{\langle t_j, DONE\rangle \mid t_j \text{ is a transition of } PN\}.$$

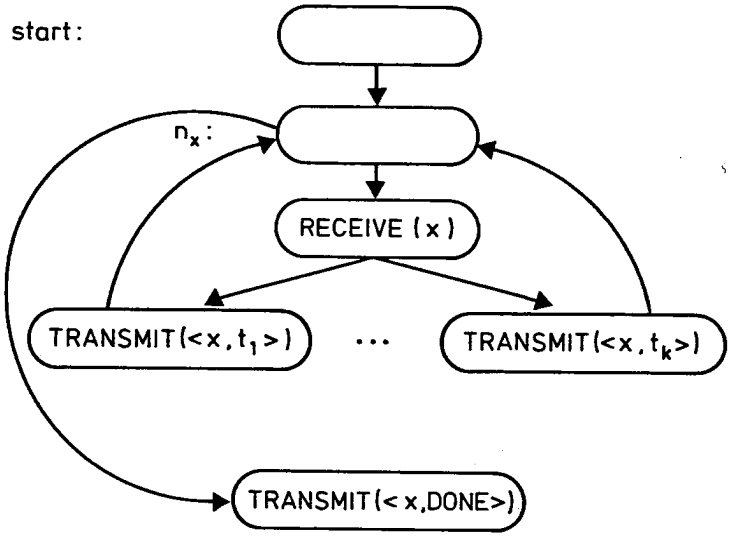
Communication is static.

For each transition $t \in T$ with input places $x_1, \ldots, x_j$ and output places $x'_1, \ldots, x'_k$, we have the following process flow graph G_t:



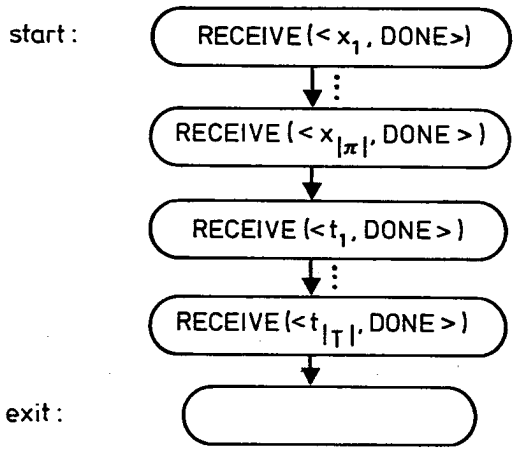
Observe that on receiving a message from each of channels $\langle x_1, t\rangle, \ldots, \langle x_j, t\rangle$, G_t transmits a message over each of channels $x'_1, \ldots, x'_k$. This process simulates the firing of transition t: the receives correspond to decrementing by one the marking of t's input places; the sends correspond to incrementing by one the marking of t's output places. Note that this simulation can go through only if the message queues of the receives were nonempty. Finally, this process can nondeterministically choose to execute its exit node (and thus transmit a message over $\langle t, DONE\rangle$) or, if possible, to simulate another firing of transition t.

For each $x \in \pi$ such that x is an input place of transitions $t_1, \ldots, t_k$, we have the following process flow graph G_x:



Observe that for each message received over channel x, G_x nondeterministically transmits over *one* of the channels $\langle x, t_1\rangle, \ldots,$ or $\langle x, t_k\rangle$. This process models the *conflict* between transitions $t_1, \ldots, t_k$ with respect to input place x: since x is shared by these transitions, the firing of any one of them may disable the other enabled transitions in this set. G_x also chooses nondeterministically whether to execute its exit node (and thus transmit a message over $\langle x, DONE\rangle$) or, if possible, to perform another simulation.

Finally, we have process flow graph G_0:



where $\pi=\{x_1, \ldots, x_{|\pi|}\}$ is the set of places and $T=\{t_1, \ldots, t_{|T|}\}$ is the set of transitions of PN. Initially, we assume that message queue $Q(x)$, $x\in\pi$, contains $\mu_0(x)$ messages. It follows from the construction that the exit node of G_0 is reached with exactly $\mu(x)$ messages in each channel $x\in\pi$ iff μ is a reachable marking in Petri net PN. The proof is straightforward and uses induction over the sequence of transition firings leading to making μ for the if direction; and induction over the execution of M_1 in which the exit node of G_0 is reached for the only if direction.

Lipton [13] has shown that the Petri net reachability and coverability problems require $2^{c\sqrt{n}}$ space, for some constant $c>0$*. He also gives a polynomial time construction of a Petri net PN_b, $b\geqq 0$, having the following properties:

- PN_b has distinguished places x, y.
- Its initial marking is μ_0, where $\mu_0(z)=0$ for all places z.
- There exists a reachable marking μ such that $\mu(y)\geqq 1$ and, for all such markings, $\mu(x)=2^b$.

By Lemma 1, this implies:

Lemma 2. *We can construct in polynomial time a set of process flow graphs with distinguished statement y and distinguished message channel x that simulate PN_b. That is, y is reachable and when reached, channel x contains always exactly 2^b messages.* □

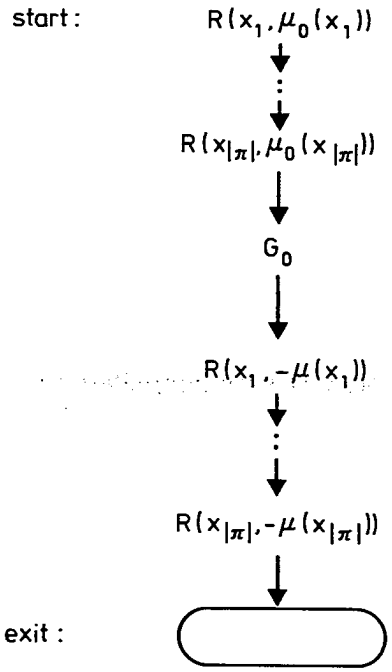
We can actually simulate PN_b using a *single* process flow graph. We will need such a flow graph to complete the proof of the following theorem.

* Reference [13] remains unpublished in a refereed format. References [14] and [23] contain published versions of Lipton's lower bound proof

Theorem 3. *Coverability in Petri nets is polynomial time reducible to reachability in our flow graph model M_1 with static communication.*

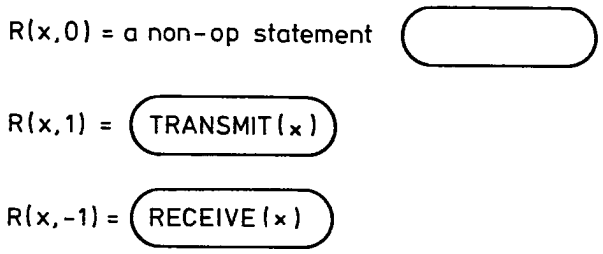
Proof. Let $PN=(\pi \cup T, E_{PN})$ be a Petri net with initial marking μ_0. We wish to test if a given marking μ is coverable in PN. Let $\pi=\{x_1, \ldots, x_{|\pi|}\}$.

The reduction uses exactly the same construction as Lemma 1, with the additional flow graph G'_0 defined as:

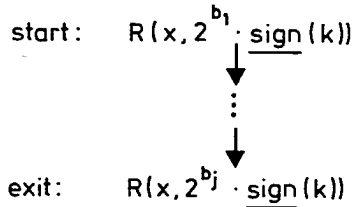


Here, for any place $x \in \pi$ and integer k, $R(x, k)$ is a subgraph which adds k messages to $Q(x)$ if $k \geqq 0$, and otherwise attempts to delete $|k|$ messages from $Q(x)$ if $k<0$. If $k<0$ and $|k|>|Q(x)|$ then the process hangs and no successors of $R(x, k)$ are ever reached. Thus, in G'_0, the first sequence of R's sets up the initial marking μ_0. G_0 then executes and arrives at some reachable marking μ'. The next sequence of R's then test if μ' covers μ. The exit node of G'_0 is reachable iff marking μ is coverable.

To complete the proof, we require a polynomial time procedure for constructing an $R(x, k)$ of size polynomial in the binary representation of $|k|$. Let



If $|k|>1$ and $|k|$ is not a power of two, then let $R(x, k)=$



where $|k|=2^{b_1}+\ldots+2^{b_j}$ is the binary expansion of $|k|$, and $\text{sign}(k)=1$ if $k\geqq 0$, else $\text{sign}(k)=-1$.

In the case that $k=2^b$, $b\geqq 1$, $R(x,k)$ is the flow graph implied by Lemma 2. □

By the lower bounds on Petri net coverability of Lipton [13], we have:

Corollary 2. *Reachability in the flow graph model M_1 with static communication is hard for* SPACE$(2^{c\sqrt{n}})$, *for some constant c.*

4. Complexity of Reachability in M_2

In this section, we analyze the complexity of testing reachability in the flow graph model M_2, where messages are copied rather than deleted from the message queues by receive statements (Sect. 2.2). Even in this simplified model and with the static communication assumption in effect, we are able to show that the problem is NP-complete.

We begin by characterizing formally the executions of M_2 in a language-theoretic framework. Let Σ be an alphabet and w an arbitrary string (not necessarily a member of Σ^*). We write $w\backslash\Sigma$ to denote the string derived from w by deleting all symbols not in Σ. Moreover, we say that a string $v\in\Sigma^*$ *is consistent with* w if $v=w\backslash\Sigma$.

Let N be a set of program statements. We demonstrate a regular language L such that the set of executions in M_2 (considered strings over N) is exactly the set of strings consistent with L. To construct L, we use a class of extended regular expressions called "regular path expressions", which are a restriction of the path expressions of Habermann [7].

A *regular path expression* is an expression α built from alphabet Σ, monadic operator *, and binary operators $+$, $\cdot$, and $\|$. $L(\alpha)$, *the language of* α is defined just as if it were a regular expression ($+$, $\cdot$, * denote the usual union, concatenation and closure operations, respectively, on strings), except for the case $\alpha=\alpha_1\|\alpha_2$. Let $\alpha_1\in\Sigma_1^*$, $\alpha_2\in\Sigma_2^*$ such that $\Sigma_1, \Sigma_2\subseteq\Sigma$. Then:

$$w\in L(\alpha) \quad \text{iff } w\backslash\Sigma_1\in L(\alpha_1) \text{ and } w\backslash\Sigma_2\in L(\alpha_2).$$

Note that if Σ_1 and Σ_2 are disjoint, then $L(\alpha_1\|\alpha_2)$ is just the arbitrary interleaving of pairs of strings in $L(\alpha_1)$ and $L(\alpha_2)$. Also, if $\Sigma_1=\Sigma_2$, then $L(\alpha_1\|\alpha_2)=L(\alpha_1)\cap L(\alpha_2)$. Intuitively, we can view α_1 and α_2 as processes, and elements of $\Sigma_1\cap\Sigma_2$ as events that must occur in synchrony by the two processes. Events not in this intersection can occur autonomously. The $\|$ operator is associative and commutative, a fact we will use below to arbitrarily extend the arity of $\|$.

We can show that the language of a regular path expression is regular by induction on its structure. For example, if $\alpha=\alpha_1\|\alpha_2$, then $L(\alpha)=\pi_1^{-1}(L(\alpha_1))\cap\pi_2^{-1}(L(\alpha_2))$. Here, π_1 and π_2 are homomorphisms defined by $\pi_i(a)=a$ for $a\in\Sigma_i$ and $\pi_i(a)=\varepsilon$ otherwise (ε denotes the empty string), $i=1, 2$. Since the regular sets are closed under intersection and under inverse homomorphism, $L(\alpha)$ is a regular set.

Let $\{P_1, \ldots, P_m\}$ be our set of communicating processes with corresponding flow graphs $\{G_1, \ldots, G_m\}$. Recall that since each process P_i is sequentially execut-

ed, an execution of P_i consists of a total ordering $(e_1, \ldots, e_k)$ such that there is a path $(n_1, \ldots, n_k)$ in $G_i=(N_i, E_i, s_i)$ beginning at its start node s_i, and e_i is the event of executing statement n_i, for $i=1, \ldots, k$.

To specify all sequential executions of P_i, let PATHS(i) be a regular expression whose language is simply the set of paths in G_i beginning at its start node s_i. For general flow graphs, PATHS(i) can be computed in time $O(|N_i|^2+|E_i|)$ and, if G_i is derived from a well-structured program (i.e., the flow graphs are reducible), then this computation is reduced to time almost linear in $|N_i|+|E_i|$. Tarjan [26, 27] gives a comprehensive description of such path problems and their solution.

To describe the executions of model M_2, we define the regular path expression:

$$\alpha_{\text{exec}} = \alpha_{\text{sequence}} \| \alpha_{\text{transmit}} \| \alpha_{\text{receive}}$$

where

$$\alpha_{\text{sequence}} = \|_i \text{ PATHS}(i)$$

where $i \in \{1, \ldots, m\}$

$$\alpha_{\text{transmit}} = \|_t (t \cdot \bar{r}_1 \cdot \ldots \cdot \bar{r}_l)^*$$

where $t \in N$ is a transmit statement over channel c and $r_1, \ldots, r_l$ are the receive statements over c

$$\alpha_{\text{receive}} = \|_r \bar{r} \cdot (\bar{r}+r)^*$$

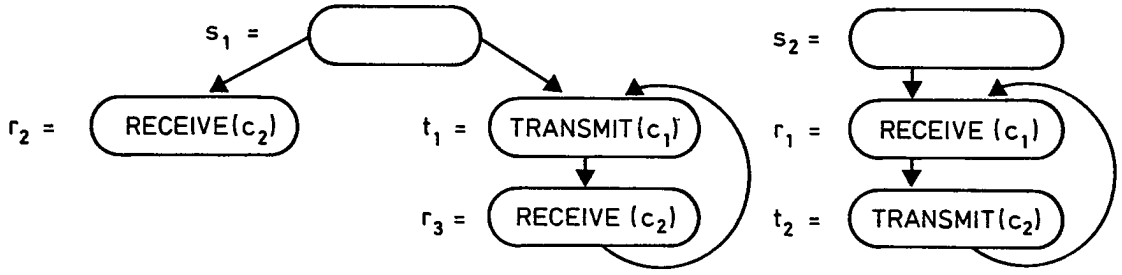
where $r \in N$ is a receive statement

The symbol $\bar{r}$ indicates that there exists a message in the appropriate message channel for the receive statement r to receive. Intuitively, α_{sequence} describes the parallel execution of the paths in the distinct flow graphs; α_{transmit} insures that a message in channel c appears only after some message has been transmitted over c; and α_{receive} insures that a receive statement on channel c takes place only if there has been at least one previous tansmission over c.

Figure 4 presents an example pair of flow graphs and their path expressions.

Theorem 4. *The set of executions of M_2 with static communication are exactly the sequences of events consistent with $L(\alpha_{\text{exec}})$.*

Proof. We first show that if w is an execution of M_2, then we can generate a string $w' \in L(\alpha_{\text{exec}})$ such that $w = w' \backslash N$. In fact, the structure of w dictates which strings we must use from $L(\alpha_{\text{sequence}})$, $L(\alpha_{\text{transmit}})$, and $L(\alpha_{\text{receive}})$. If $w \backslash N_i = p_i$ then we must choose p_i from $L(\text{PATHS}(i))$, $1 \leqq i \leqq m$. If in w there are k_t occurrences of transmit statement t, then we must choose the string $(t \cdot \bar{r}_1 \cdot \ldots \cdot \bar{r}_l)^{k_t}$ from $L(\alpha_{\text{transmit}})$. Finally, let there be k_r occurrences of receive statement r in w. Assuming that r names c and there are k_c transmit statements in w naming c, then we must choose a string from w that begins with an $\bar{r}$ and contains k_c $\bar{r}$'s and k_r r's. We now observe that these three strings may be interleaved arbitrarily with the proviso that any r event is preceded in w' by at least one corresponding t event. By the operational semantics of M_2, this constraint must



$$\text{PATHS}(1) = \text{prefix-closure}[s_1 \cdot (r_2 + (t_1 \cdot r_3))^*]$$

$$\text{PATHS}(2) = \text{prefix-closure}[s_2 \cdot (r_1 \cdot t_2)^*]$$

$$\alpha_{\text{sequence}} = \text{PATHS}(1) \| \text{PATHS}(2)$$

$$\alpha_{\text{transmit}} = (t_1 \cdot \bar{r}_1)^* \| (t_2 \cdot \bar{r}_2 \cdot \bar{r}_3)^*$$

$$\alpha_{\text{receive}} = \|_{i=1}^{3} \bar{r}_i \cdot (\bar{r}_i + r_i)^*$$

Fig. 4. Example pair of flow graphs and their path expressions

also be present in w. Thus we have enough leeway to generate a string w' with which w is consistent, for any legal execution w.

We are left to show that if $w \in L(\alpha_{\text{exec}})$, then there exists an execution w' of M_2 that is consistent with w, i.e., $w' = w \backslash N$. The constraint on w imposed by α_{sequence} ensures that $w \backslash N_i \in L(\text{PATHS}(i))$, i.e., $w \backslash N_i$ is a sequential execution of P_i, $1 \leqq i \leqq m$. The combination of α_{transmit} and α_{receive} ensures that any receive statement r over channel c is preceded in w by at least one transmit statement t over channel c. Thus $w \backslash N$ is a legal execution of M_2. □

Applying Theorem 4, we can characterize the reachability of a program statement n relative to model M_2 in terms of the nonemptiness problem for regular path expressions. Let P_i be the process containing n and let $G_i = (N_i, E_i, s_i)$ be its program flow graph. Let $\text{PATHS}'(i) \subseteq \text{PATHS}(i)$ be the regular expression whose language is the set of paths in G_i starting at s_i and containing program statement n. Let α'_{exec} be the regular path expression defined just like α_{exec}, except that $\text{PATHS}'(i)$ is substituted for $\text{PATHS}(i)$. Then by Theorem 4:

Corollary 3. *$L(\alpha'_{\text{exec}}) \neq \emptyset$ iff n is reachable in M_2.*

We now consider the complexity of reachability in M_2 and, equivalently, the complexity of the nonemptiness problem for regular path expressions. Surprisingly, α'_{exec} has sufficiently restricted structure that we have a nondeterministic polynomial time algorithm for testing nonemptiness of $L(\alpha'_{\text{exec}})$.

Theorem 5. *Reachability in M_2 with static communication is* NP*-complete.*

Proof. First we present a nondeterministic polynomial time algorithm for testing reachability. Informally, it suffices to show that if a program statement n is reachable at all, it is reachable within a short (i.e., polynomial in $|N|$) execution. We call such an execution a "witness" to the reachability of n.

Formally, a path in the program flow graph G_i of a process P_i is *acceptable relative to an execution w* iff: (i) it begins either at an immediate successor of a statement appearing in w, or at the start node of G_i if no statements of G_i appear in w; and (ii) the path contains no receive statements over a channel on which in w there have been no previous transmissions.

To construct a witness execution w for n, we put forth the following algorithm:

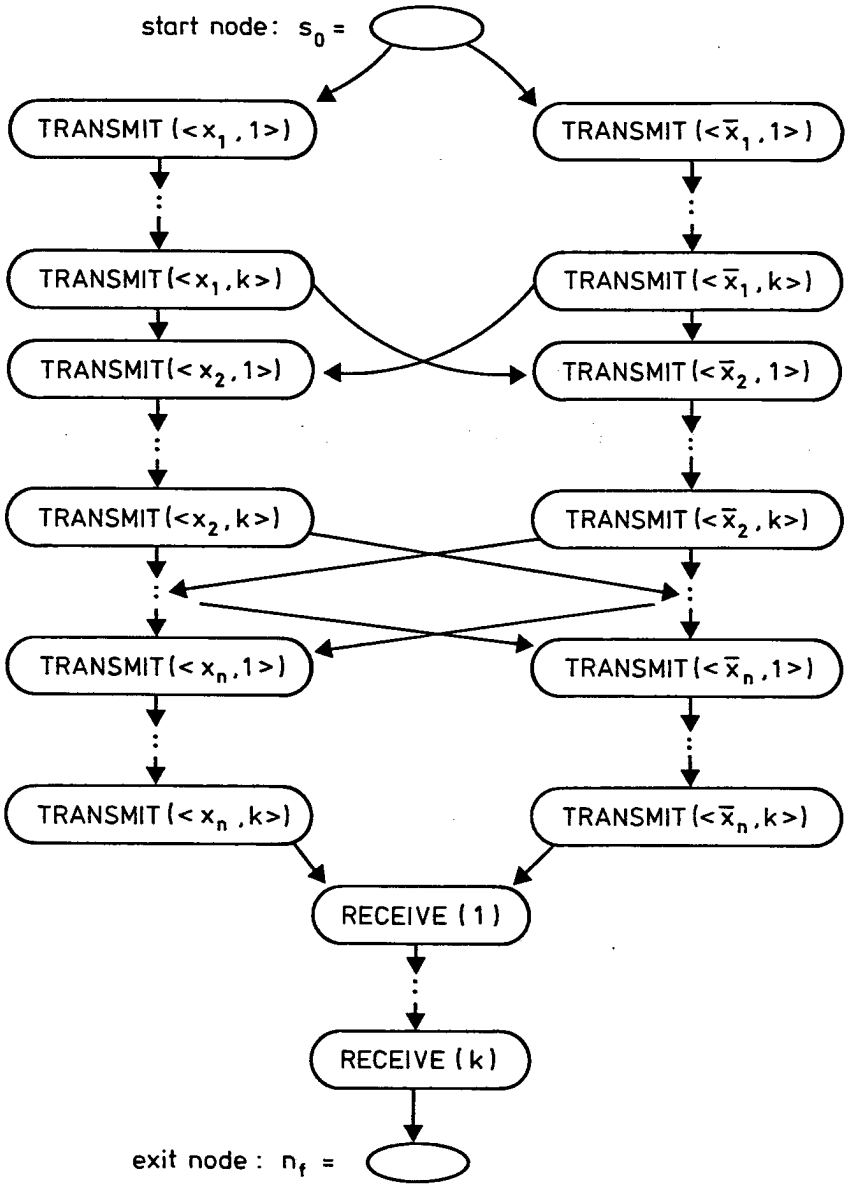
(1) Initially, w is empty.
(2) If there is an acceptable path to n relative to w, then append this path to w and output "n is reachable".
(3) Otherwise, nondeterministically choose a transmit statement m over a channel with no previous transmissions in w, and such that there is an acceptable path p to m. If no such path exists, then output "n is unreachable".
(4) Append this path p to w and go to (2).

The time required by this algorithm is linear in $|N|$ since for each process P_i with flow graph G_i, P_i's contribution to w corresponds to a cycle-free path in G_i.

Next we show that testing reachability is NP-hard using a reduction from satisfiability of boolean formulas in 3-conjunctive normal form [2].

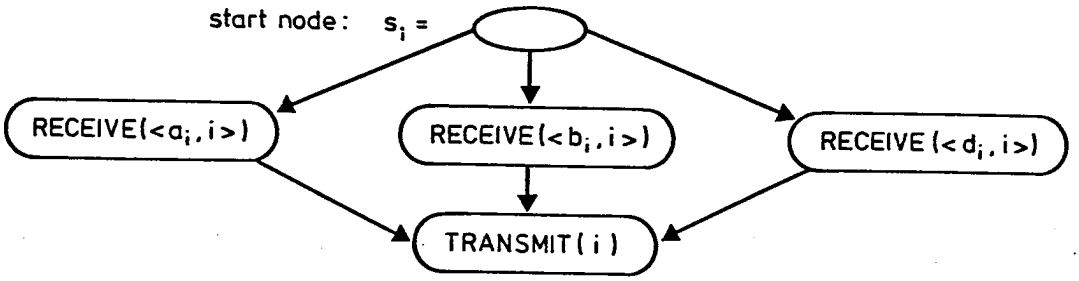
Let $\{X_1, \ldots, X_n\}$ be a set of n boolean variables. Let $B = C_1 \wedge \ldots \wedge C_k$ be a boolean formula with k clauses $C_i = (a_i \vee b_i \vee d_i)$, where a_i, b_i, d_i are literals in $\{X_1, \ldots, X_n, \bar{X}_1, \ldots, \bar{X}_n\}$.

We construct $k+1$ processes with program flow graphs $G_0, G_1, \ldots, G_k$, where $G_0 = (N_0, E_0, s_0)$ has the form.



Observe that from its start node, G_0 nondeterministically chooses to transmit messages over channels $\langle X_1, 1\rangle, \ldots, \langle X_1, k\rangle$, or over channels $\langle \bar{X}_1, 1\rangle, \ldots, \langle \bar{X}_1, k\rangle$. Intuitively, the former choice corresponds to a truth assignment in which variable X_1 appears unnegated, while the latter choice corresponds to a truth assignment in which X_1 appears negated. After these k transmissions, G_0 is then faced with the same type of nondeterministic choice for $X_2, \ldots, X_n$. After completing the truth assignment, G_0 waits to receive messages in succession from channels $1, \ldots, k$. As we will see, G_0's successful completion of these k receives will correspond to it having chosen a satisfying truth assignment for B.

For each clause C_i, $1 \leqq i \leqq k$, we have a program flow graph $G_i=(N_i, E_i, s_i)$:



It is easy to verify from the above that the exit node n_f of G_0 is reachable iff $B=C_1 \wedge \ldots \wedge C_k$ has some satisfying truth assignment. For the if direction, let $V=\{Y_1, \ldots, Y_n\}$ be a satisfying truth assignment for B. Consider the execution where G_0 branches left to perform its ith sequence of transmissions to the clause processes when $Y_i=X_i$, and branches right when $Y_i=\bar{X}_i$ (G_0 is totally free to do so). Since V is a satisfying truth assignment for B and messages in M_2 are only copied rather than deleted from message queues, these choices made by G_0 will enable each of the clause processes C_j to reach its transmit node. The execution of these k transmit nodes (which are always enabled) will in turn ensure the reachability of G_0's exit node.

For the only if direction, assume that the exit node of G_0 is reachable. Then, by the operational semantics of M_2, the transmit node of each clause process C_j must be reachable. This in turn implies the satisfiability of B. □

5. Conclusions

We have considered flow graph models M_1 and M_2 of communicating processes. Our complexity results indicate that analysis problems in these models, such as reachability, require high computational effort but are at least decidable in the case of static communication. We believe that the models M_1 and M_2 are nevertheless interesting because of the relationship we have shown between them and Petri nets and Habermann's path expressions.

In [28] and [11], the potential deadlock problem for systems of processes that communicate synchronously was shown to be computationally intractable. By restricting the structure of processes or the way in which they are interconnected, several interesting polynomial subcases of the problem were identified [11]. It would be interesting to see if similar results could be obtained in the decidable models of asynchronous processes presented in this paper.

Acknowledgements. The authors are indebted to the anonymous referees for their many constructive comments, and for pointing out several references.

References

1. Brand, D., Zafiropulo, P.: On communicating finite state machines. J. ACM **30**, 323–342 (1983)
2. Cook, S.A.: The complexity of theorem-proving procedures. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151–158. New York: ACM 1971
3. Cunha, P.R.F., Maibaum, T.S.E.: A synchronization calculus for message oriented programming. Proceedings of the 2nd International Conference on IEEE Computer Society Press Distributed Computing Systems, Paris, France. pp. 433–445. Washington, D.C. 1981
4. Feldman, J.A.: A programming methodology for distributed computing (among other things). Commun. ACM **22**, 353–368 (1979)
5. Gouda, M.G., Gurari, E., Lai, T., Rosier, L.: On deadlock detection in systems of communicating finite state machines. Comput. Artif. Intell. **6**, 209–228 (1986). Also available as Technical Report TR-84-11, Department of Computer Science, University of Texas at Austin (1984)
6. Gouda, M.G., Rosier, L.: Priority networks of communicating finite state machines. SIAM J. Comput. **14**, 569–584 (1985)
7. Habermann, A.N.: Path expressions. Carnegie-Mellon Univ. (1975)
8. Hecht, M.S.: Data flow analysis of computer programs. New York: American Elsevier 1977
9. Hoare, C.A.R.: Communicating sequential processes. Comm. ACM **21**, 666–677 (1978)
10. INMOS Limited. Occam Programming Manual. London: Prentice-Hall 1984
11. Kanellakis, P.C., Smolka, S.A.: On the analysis of cooperation and antagonism in networks of communicating processes. Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada. pp. 23–38. New York: ACM 1985
12. Ladner, R.: The complexity of problems in systems of communicating processes. J. Comput. Syst. Sci. **21**, 179–194 (1980)
13. Lipton, R.: The reachability problem requires exponential space. Research Report 62, Department of Computer Science, Yale University, New Haven, CT (1976)
14. Mayr, E., Meyer, A.: The Complexity of the word problems for commutative semigroups and polynomial ideals. Adv. Math. **46**, 305–329 (1982)
15. Milner, R.: A Calculus of Communicating Systems. In: Lecture Notes in Computer Science. Vol. 92. Berlin Heidelberg New York: Springer 1980
16. Minsky, M.L.: Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. Ann. Math. **74**, 437–455 (1961)
17. Pachl, J.K.: Reachability problems for communicating finite state machines, Research Report CS-82-12, Dept. Computer Science, Univ. Waterloo, Waterloo, Ontario, Canada (1982)
18. Peterson, J.L.: Petri net theory and the modeling of systems. Englewood Cliffs: Prentice-Hall Inc. 1981
19. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. **6**, 223–231 (1978)
20. Rauchle, T., Toueg, S.: Exposure to deadlock for communicating processes is hard to detect. Inform. Process. Lett. **21**, 63–68 (1978)
21. Reif, J.H., Smolka, S.A.: Reachability and data flow analysis of distributed communicating processes: a unified approach. (Submitted for publication)
22. Reif, J.H., Spirakis, P.: Distributed algorithms for synchronizing interprocess communication within real time. Proceedings of the 13th ACM Symposium on Theory of Computation, Madison, WI. pp. 133–145. New York: ACM 1981. Also rewritten as: Real-time synchronization of interprocess communications. Technical Report TR-25-82, Aiken Computation Lab, Harvard Univ., Cambridge, MA (1982)
23. Rosier, L., Yen, H.C.: A multiparameter analysis of the boundedness problem for vector addition systems. J. Comput. Syst. Sci. **32**, 105–135 (1986)
24. Rosier, L., Yen, H.C.: Boundedness, empty channel detection, and synchronization for communicating finite automata. Theor. Comput. Sci. **44**, 60–105 (1986)
25. Strom, R.E., Halim, N.: A new programming methodology for long-lived software systems. IBM J. Res. Devel. **28**, 52–59 (1984)

26. Tarjan, R.E.: A unified approach to path problems. J. ACM **28**, 577–593 (1981)
27. Tarjan, R.E.: Fast algorithms for solving path problems. J. ACM **28**, 594–614 (1981)
28. Taylor, R.N.: Complexity of analyzing the synchronization structure of concurrent programs. Acta Informatica **19**, 57–84 (1983)
29. Yu, Y., Gouda, M.: Unboundedness detection for a class of communicating finite state machines. Inform. Process. Lett. **17**, 235–240 (1983)

Note

ACM = The Association for Computing Machinery, Inc.
IEEE = The Institute of Electrical and Electronics Engineers, Inc.

Received August 13, 1986 / January 12, 1988