TERMINATION OF PROBABILISTIC CONCURRENT PROGRAMS
(Extended Abstract)

Sergiu Hart, Micha Sharir
Tel Aviv University

Amir Pnueli
Weizmann Institute of Science

Abstract.

The asynchronous execution behavior of several concurrent processes, which may use randomization, is studied. Viewing each process as a discrete Markov chain over the set of common execution states, we give necessary and sufficient conditions for the processes to converge almost surely to a given set of goal states, under any fair, but otherwise arbitrary schedule, provided that the state space is finite. (These conditions can be checked mechanically.) An interesting feature of the proof method is that it depends only on the topology of the transitions and not on the actual values of the probabilities. We also show that in our model synchronization protocols that use randomization are in certain cases no more powerful than deterministic protocols. This is demonstrated by (a) Proving lower bounds on the size of a shared variable necessary to ensure mutual exlusion and lockout-free behavior of the protocol; and (b) Showing that no fully symmetric 'randomized' protocol can ensure mutual exclusion and freedom from lockout.

0. Introduction.

Randomization has proven to be a very useful tool in the construction of certain algorithms for parallel processes, which behave 'better' than their deterministic counterparts, by using less shared memory, having relatively simpler structure, and in certain cases accomplishing goals that deterministic algorithms provably can not accomplish. The price that one usually has to pay in using such a probabilistic algorithm is that certain properties that are required from the algorithm will happen with probability 1, but not necessarily with certainty. Recently published algorithms of this sort include synchronization protocols [Ra1], choice coordination [Ra2], a symmetric distributed solution to the dining philosophers' problem [LR], a probabilistic implementation of a common resource allocation scheme [FR], and various algorithms in symmetric distributed systems [IR].

The problem that one encounters in designing such an algorithm is in showing that the algorithm is correct in a probabilistic sense. This is quite a tedious task, since one must consider all possible sequences of execution steps taken by the processes and show that none of them can prevent a certain desired event from happening with probability 1. To quote Lehmann and Rabin in [LR],

"The realm of proofs of correctness for concurrent processes is not well known. As the reader will realize, proofs of correctness for probabilistic distributed systems are extremely slippery."

Our aim in this paper is to shed some light on this unexplored realm. In particular, assuming that each process has only finitely many states, we will give necessary and sufficient conditions for certain probabilistic properties of a parallel algorithm to hold with probability 1. Moreover, these conditions can be tested algorithmically, given the "atomic" structure of each process. That is, we present a fully mechanical decision procedure of certain probabilistic properties, including, e.g., freedom from deadlock and from lockout.

We will also be concerned with comparing the capabilities of probabilistic synchronization protocols with those of deterministic protocols. We will provide in Section 2 two instances in which the probabilistic approach is provably not more powerful than the deterministic approach. These instances are: (a) Achieving mutual exclusion and freedom from individual lockout by using a shared variable v on which indivisible 'test-and-set' operations are allowed (here the size of v must be $\Omega(N)$ in both cases, where N is the number of processes). (b) Achieving those properties by a fully symmetric protocol which uses a shared

variable on which only indivisible read and write operations are allowed. (This is impossible to achieve in either approach.)

It turns out that a major conceptual problem in the analysis of a probabilistic parallel program is the choice of the right model for the possible execution sequences of the processes. It is generally accepted paradigm that the processes can be viewed as executing in sequence (with no two processes ever executing simultaneously). At each step some process is chosen to perform the next 'atomic' action. The decisions which process will be chosen to operate next are thought of as being taken by some imaginary scheduler, and the main problem is: what rules can the scheduler use in its decisions. For example, the admissible schedules considered by Rabin in [Ra1] are more restricted than those considered by Lehmann and Rabin in [LR]. We will show, as a consequence of our general theorems, that Rabin's algorithm for synchronization given in [Ra1] does not have the required properties if one allows also schedules of the sort considered in [LR].

1. Characterization of Probabilistic Concurrent Termination.

The model that we use for representation of a probabilistic parallel program is the following: States in the program are identified by the location in each process and by the values of shared variables and local variables. With each process k we associate a transition probability matrix P^k describing the possible state transitions that can occur as a result of a single atomic action of k, i.e., P^k_{ij} is the probability of reaching state j from state i under one step of process k. Let I denote the set of all possible states. Let $P^1,\ldots,P^s$ be the transition matrices of the processes $1,\ldots s$ participating in such an asynchronous parallel program. An execution tree, also referred to as a schedule σ, is a tree constructed as follows: Nodes in the tree are pairs (i,k) where $i \in I$ is a program state and $k \in \{1,\ldots,s\}$ is the process scheduled to operate on that node. The root is labeled by (i_0,k) where i_0 is the initial state. The edges going out of a node labeled by (i,k) correspond to nonzero entries in the i-th row of P^k, and designate transitions having positive probabilities from state i to other states under execution of process k. Thus an edge from (i_1,k_1) to (i_2,k_2) means that if process k_1 is scheduled to operate on the state i_1, there is a positive probability to reach state i_2. The appearance of these nodes in the schedule σ implies that k_1 is indeed scheduled at i_1 and k_2 at i_2 if the execution actually gets there. Note that the scheduler's decisions may depend on the whole path in the tree leading to a node, and so different processes may be scheduled on the same state in different nodes of the tree.

To ensure proper functioning of the parallel program, we will require that σ possesses certain "fairness" properties.

Definition. A schedule σ is called strictly fair if each path Π in the associated transition tree is a fair path, i.e. each process $k \in K$ labels infinitely many nodes of Π.

That is, in each execution of the program each process is scheduled infinitely often. This is a standard requirement (see [LR] for example, where this is called "proper") in analysis of parallel programs.

We will also consider a weaker notion of fairness. For this, note that a schedule σ induces a probability measure $\mu = \mu_\sigma$ on the sequence space whose elements are paths in the transition tree associated with σ. This measure is defined on the Borel field generated by all the finite cylinders (i.e. sets of paths having the same initial segment) such that the measure of such a cylinder is the product of probabilities of the edges of the common initial path.

Definition. A schedule σ is called fair if μ_σ-almost every path in the associated transition tree is fair.

As will be shown below, fair schedules are in some sense limit points of strictly fair schedules, so that with no loss of generality, we could deal with either class of schedules, and obtain essentially the same results.

The main problem that we shall be concerned is the probability of reachability of certain states from other states. For example, if i is a state in which a process k is trying to enter its critical section, and X is the set of all states in which k is at its critical section, then the probability of reaching X from i is the probability that k will not be locked out of the set X at state i. Similarly, if X_1 is the set of states in which some process is at its critical section, then the probability of reaching X_1 from i is the probability that the system will not be deadlocked at state i.

Let therefore $X \subset I$ be a given set of 'goal' states, and let $i \in I$ be an 'initial' state. Our aim is to compute

$$f^*_{i,X}(\sigma) \equiv \text{probability of ever reaching a state in } X \text{ from } i \text{ under } \sigma ,$$

where σ is some fair schedule. Note that this can also be expressed as

$$f^*_{i,X}(\sigma) = \sum_{n=0}^{\infty} f^{(n)}_{i,X}(\sigma) ,$$

where

$$f^{(n)}_{i,X}(\sigma) = \text{probability of reaching a state in } X \text{ from } i \text{ for the first time after exactly } n \text{ steps under } \sigma .$$

Since we will be interested in showing that $f^*_{i,X}(\sigma) = 1$ for every fair schedule σ, we define

$$h^*_{i,X} = \inf\{f^*_{i,X}(\sigma) : \sigma \text{ a fair schedule}\}$$

and we seek conditions under which $h^*_{i,X} = 1$.

Let us first explain our reasoning intuitively. The way in which an adversary scheduler can prevent the system from ever entering X is to iterate forever through a set of states E disjoint from X, scheduling at each $i \in E$ some process k which transfers i only to states in E, and using all processes in this manner. We will show that the nonexistence of such a set E is a necessary and sufficient condition for $h^*_{i,X}$ to be 1.

We begin by a precise definition of the above notion:

Definition. A set $E \subset I$ is called K-ergodic if the following holds: For each $i \in E$ define

$$K_E(i) \equiv \{k \in K \mid P^k_{i,E} \equiv \sum_{j \in E} P^k_{ij} = 1\}$$

(this is the set of processes that cannot lead from i to a state outside E). Then we require that

(i) $\bigcup_{i \in E} K_E(i) = K$, i.e. for every process $k \in K$ there is a state $i \in E$ such that all k transitions out of i are still in E.

(ii) For every $i \neq j \in E$ there exists a chain $i_0, i_1, \ldots, i_r \in E$ with $i_0 = i$, $i_r = j$, and a chain $k_0, k_1, \ldots, k_{r-1} \in K$ of processes such that for each $s = 0, 1, \ldots, r-1$

$$k_s \in K_E(i_s)$$

and

$$P^{k_s}_{i_s i_{s+1}} > 0$$

(a set E satisfying (ii) will be called communicating).

Let $\hat{i} \in I$ be the initial state, and $X \subset I$ be the goal set with $\hat{i} \notin X$. Define $\hat{I}$ as the set of all states that can be reached (with positive probability) from $\hat{i}$ before a state in X is reached using any finite sequence of processes. $\hat{I}$ includes $\hat{i}$ and is disjoint from X. Our main result is

Theorem 1: Let $\hat{i}$, X, $\hat{I}$ be as above, and assume that $\hat{I}$ is finite. Then the following conditions are equivalent:

(1) $h^*_{\hat{i},X} = 1$

(2) $h^*_{i,X} = 1$, for each $i \in \hat{I}$.

(3) $\hat{I}$ does not contain any K-ergodic set

(4) There exists a decomposition of $\hat{I}$ into disjoint sets $I_1, \ldots, I_n$, with $I_0 = X$, such that if we put $J_m = \bigcup_{r=0}^{m} I_r$, $m = 0,1,\ldots,n$, then for each $m = 1,2,\ldots,n$ we have

(a) For each $i \in I_m$, $k \in K$,

$$P^k_{i,J_{m-1}} = 0 \Rightarrow P^k_{i,I_m} = 1$$

(b) There exists $k \equiv k(m) \in K$ such that for each $i \in I_m$ $P^k_{i,J_{m-1}} > 0$.

(Condition (4.a) says that if process k can transfer the system from a state in I_m to a state outside I_m, then some k-transitions (with non-zero probabilities) move the system 'down' the chain I_r, towards the goal I_0; Condition (4.b) ensures the existence of at least one process that would do this for all states in I_m.)

The proof is technical and involved, and is therefore omitted in this version. Part of the proof, namely that (3) is equivalent to (4), is implied by the algorithm below.

Remarks.

(1) We can regard the partition of I into $I_1, \ldots, I_n$ as the assignment of an element of the well founded set $\{1,\ldots,n\}$ to each state $i \in I$. That is, each $i \in I_m$, $1 \leq m \leq n$ is assigned the value m. Denote this assignment by $\rho: I \to \{1,\ldots,n\}$.

Condition (4.a) then states that for every i and k, if there is no k transition leading to i' such that $\rho(i) > \rho(i')$, then all k transitions must lead to i's such that $\rho(i) = \rho(i')$.

Condition (4.b) states that for every m, $1 \leq m \leq n$ there exists a $k = k(m)$ such that for every $i \in I_m$, there is a k transition from i to i' with $\rho(i') < \rho(i) = m$.

This view shows our method to be an extension of the proof method given in [LPS] for the fair termination of deterministic concurrent programs. Indeed, if we restrict ourselves to deterministic processes (i.e., assume that for any $i \in I$, $k \in K$, $P^k_{ij} = 1$ for exactly one $j \in I$), then our condition 4 reduces to the proof method given in [LPS]. An open question is whether our results can be extended to processes with infinite state sets.

(2) An immediate corollary of the preceding theorem is that the convergence is independent of the particular values of nonzero transition probabilities.

(3) It is also easy to show that, under the assumptions of the preceding theorem, $\min_{i \in \hat{I}} h^*_{i,X}$ is either 0 or 1. We also show that the preceding theorem remains true if one admits only strictly fair schedules.

Next we present an algorithm that, given the set $\hat{I}$ of intermediate states, the set X of goal states, and the transition probability matrices of the processes, either constructs a K-ergodic set $E \subset \hat{I}$, thereby showing that $h^*_{\hat{i},X} < 1$, or else builds up the decomposition $\{I_m\}$ as in the statement of the theorem, thereby showing that $h^*_{\hat{i},X} = 1$.

The algorithm proceeds as follows:

(a) Construct a transition graph G, which is a directed labeled graph whose nodes are elements of $\hat{I}$, with one additional node designating X; for each $k \in K$, each $i \in \hat{I}$ and $j \in \hat{I}$ (or $j = X$) such that $P^k_{i,j} > 0$, we draw an edge e from i

to j and label e by k.

(b) We begin to construct a decomposition $\{I_m\}$ by putting $I_0 = X$.

(c) Suppose that $I_0,\ldots,I_r$ have already been constructed. Delete from G all nodes belonging to the union $J_r = \bigcup_{k=0}^{r} I_k$ of these sets; also for each $i \in \hat{I} \setminus J_r$ delete from G all edges e such that either e leads from i to some $j \in J_r$, or there exists another edge e' leading from i to some node in J_r, and e and e' are labeled by the same process $k \in K$. (In other words, we ignore states that have already been assigned a rank, and transitions from a state i by processes that can, in a possibly alternative transition from state i, move the system into J_r.) Let G_r denote the resulting graph.

(d) Partition G_r into strongly connected components, and let E be such a component having no successors (terminal component). If each process $k \in K$ labels some internal edge of E, then E is a K-ergodic set, in which case the algorithm halts, having found such a set; otherwise, put $I_{r+1} = E$, define $k(r+1)$ to be a process k which does not label any internal edge of E, and repeat steps (c) and (d).

(e) If finally G empties out, then we have found the desired decomposition of I.

An example. Consider the following two-process synchronization, where a shared variable c is used (we assume indivisible 'test and set' operations on c). c has 3 values: 0 - designating a neutral state, 1 - a state in which process k_1 will enter its critical section, and 2 - where k_2 will enter. The code for k_1 (resp. k_2) is as follows:

```
Try:  If  c = 1  [resp. 2] then go to  Ex
      else
          if  c = 0  then  c:=Random (1,2)
              fi; /* draw 1 or 2 with
              equal probabilities */

          go to Try
       fi

Ex : - critical region -
     c := 0
     go to Try
```

Here we have 5 states, each represented as (c,ℓ_1,ℓ_2), where c is the value of the shared variable, and where ℓ_1 (resp. ℓ_2) is the location in process k_1 (resp. k_2), which can assume the values T (for being at the trying section) and X (for being in the critical section). We assume that both processes remain live indefinitely. The states are

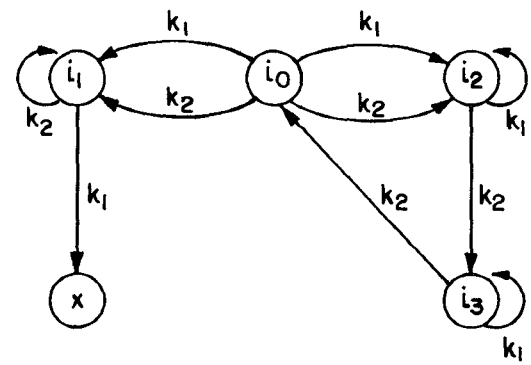
$$i_0 = (0,T,T)$$
$$i_1 = (1,T,T)$$
$$i_2 = (2,T,T)$$
$$i_3 = (2,T,X)$$
$$X = i_4 = (1,X,T)$$

and the transition matrices are

	i_0	i_1	i_2	i_3	X	i_0	i_1	i_2	i_3	X
i_0		$\frac{1}{2}$	$\frac{1}{2}$				$\frac{1}{2}$	$\frac{1}{2}$		
i_1				1			1			
i_2		1							1	
i_3			1			1				

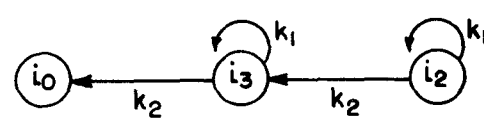
We will show that $\hat{I} = \{i_0,i_1,i_2,i_3\}$ does not contain a K-ergodic set by applying the algorithm described above. Begin by putting $I_0 = X = \{i_4\}$. The other sets are found as follows:

We first construct G:



(each edge is labeled by the process that induces it).

To find I_1, delete the k_1 - edge from i_1 to X to obtain G_0. A terminal strongly connected component in G_0 is $\{i_1\}$, and only k_2 labels internal edges of this component. Hennce, we put $I_1 = \{i_1\}$, $k(1) = k_1$. To find I_2, delete from G_0 the node i_1, and all the edges going out of i_0 (some of these edges lead to I_1, while the other edges are siblings of such edges, in the sense defined above.) G_1 thus has the following form



from which one easily obtains $I_2 = \{i_0\}$, $k(2) = k_1$, $I_3 = \{i_3\}$, $k(3) = k_2$ and $I_4 = \{i_4\}$, $k(4) = k_2$.

We emphasize again that this construction and its consequences do not depend on the particular values of nonzero transition probabilities used in the randomization statements in the processes involved.

2. Probabilistic vs. Deterministic Protocols.

In this section we compare the capabilities of probabilistic protocols with those of deterministic protocols. An example motivating this study is an algorithm, given in [Ra1], for synchronization of N processes, which uses only $0(\log N)$-valued shared variable on which indivisible test-and-set operations are allowed. On the other hand, it is shown in [BFJLP] that in any deterministic synchronization algorithm for N processes using test-and-set operations on a shared variable, and providing both mutual exclusion and freedom from lockout, the shared variable must have about $\sqrt{2N}$ values, and this lower bound increases to about $N/2$ values if certain natural constraints are imposed on the structure of the processes involved. The algorithm given in [Ra1] is shown to provide mutual exclusion and to be lockout-free. However, the schedules under which this algorithm is assumed to operate form a restricted subset of the general fair schedules that we consider. It is helpful in this regard to ignore the 'demonic' nature of a schedule, and simply interpret it as an integral part of the execution tree. Thus the schedule does not really 'use' the past history of an execution to determine the next process to be scheduled, but simply records at each execution step which process is next ready to perform an atomic action. Other families of schedules considered in the literature impose certain constraints on the scheduling, e.g. bounded speed ratios between processes [RS], or, as is the case in [Ra1], allowing the schedule to 'base its decisions' only upon partial information concerning past execution history, so that it must make the same scheduling choice for all sequences having the same partial structure. Such a restriction on the behavior of the schedule can improve the performance of a synchronization algorithm, but may make it less robust, in the sense that it may fail to meet some of its requirements if more general (fair) schedules are allowed.

We show in this section that Rabin's algorithm is not lockout-free if general fair schedules are allowed. As it turns out, this fact does not depend on the particular form of the algorithm, but follows from a general lower bound on the size of a shared variable necessary to ensure both mutual exclusion and freedom from lockout. Specifically, we show that the lower bounds given in [BFJLP] are also required for probabilistic synchronization algorithms of the same kind. This will follow from a generalization of the proofs given in [BFJLP] to the probabilistic case. Thus any algorithm attempting to use fewer values for such a shared variable is doomed to failure against general fair schedules, although it may still be successful against 'weaker' schedules, as is indeed the case in [Ra1].

These results can be summarized as follows:

Proposition 1: The algorithm in [Ra1] is almost surely lockout-free against the restricted family of (fair) schedules introduced there, but fails to have this property if general fair schedules are allowed.

Indeed, an "adversary" scheduler may base its decisions on values obtained by random assignments. Thus a program based on the assumption that if a variable in a process is repeatedly assigned a random value, and a certain value of this choice will ensure admittance into a critical section, if scheduled immediately then eventually the process will be admitted - such a program may fail under a generally "adversary" scheduler. This is so since the scheduler may observe the randomly chosen value and schedule the process only when such scheduling will not enable the process to access its critical section. Such scheduling is still fair with probability 1.

This is not the case with Rabin's scheduler whose decisions may not depend on randomly assigned values. Consequently a program correct under Rabin's scheduler may still fail under generally fair scheduler as defined here.

Theorem 2: Suppose that N processes participate in a synchronization protocol using a shared variable v on which indivisible test-and-set operations are allowed. Then at least $\sqrt{2N}$ values of v are required to ensure both mutual exclusion and freedom from lockout. Furthermore, if each process has only a single state while being in its idle section (i.e. if it cannot remmeber any past experience), then at least $(N+1)/2$ values for v are required.

Proof: A generalization of the proofs given in [BFJLP].

The results stated so far in this section raise the general question: Under what circumstances does randomization really help us to obtain 'better' algorithms? The above negative result shows that we cannot hope to save space by incorporating randomization into synchronization protocols of the kind discussed in [BFJLP] and [Ra1] (if we admit general fair schedules). We will next give another negative results. Consider a synchronization process involving $N \geq 2$ identical processes, all of which use a common variable v which they may read from, or write into, in one indivisible step (all processes access v in exactly the same manner). It is well known that no deterministic solution to this problem, that provides mutual exclusion, can exist. Surprisingly enough, an analogous proof shows that a similar result holds in the probabilitisticcase as well:

Theorem 3: There does not exist a probabilistic fully symmetric protocol that solves the above synchronization problem, and ensures both mutual exclusion and freedom from lockout with probability 1.

Proof: Suppose that there exists such a protocol. We assume that no lockout can occur with positive probability. Hence, if we start from a symmetric configuration i in which all processes are idle, and we let only one process (say k_1) execute by itself and leave the other processes in their idle state, k_1 must eventually enter its critical section with probability 1. This implies that

there exists at least one finite execution path Π in which k_1 executes alone, and all other processes remain idle, such that all transitions on Π have positive probabilities and such that k_1 gets via Π from its idle section to its critical section. Let the product of the transition probabilities along Π be $p > 0$.

Let us now execute the protocol as follows. Start at the symmetric configuration i, and assume that all processes become active (i.e. get out of the idle section) together. Schedule processes in a round robin fashion, letting each of them perform one atomic action at its turn. We will construct a path (having positive probability) such that at the end of each round, the configuration is again symmetric. Assume that this were the case at the beginning of some round r. Then each process is at the same internal state and so performs the same action in its turn. If this action does not involve randomization, then either all the processes in this round manipulate their internal variables in the same way, or they all read the (same value of the) shared variable, or they all write the same value into the shared variable. Hence, in either case, their final configuration is again symmetric. Note that in this mode of scheduling, each process is not aware that other processes are also executing, so that each process will follow a sequence of states identical to those that k_1 passed by itself along the sequence Π (as long as there is not randomization) Now suppose that the common action of the processes in round r did involve randomization. Then with some positive probability $p_r > 0$, k_1 will make the choice that will keep it in the sequence Π, and with the same probability each other process will make an identical choice. Consequently, with probability p_r^N (where N is the number of processes) all processes would make this common choice, and will again reach a symmetric configuration; moreover, in this new configuration each process is unaware of the existence of the other processes, and its state is the symmetric image of the r-th state of k_1 on Π.

Continuing in this manner, with probability $p^N > 0$ all processes would enter their critical sections together after Π steps, since k_1 does so and the behavior of the other processes remains symmetric to that of k_1 under this particular execution sequence. This contradicts our assumptions, and so proves the theorem.

Q.E.D.

REFERENCES

[BFJLP] J. E. Burns, M. J. Fischer, P. Jackson, N. A. Lynch and G. L. Peterson, "Shared Data Requirements for Iplementation of Mutual Exclusion Using a Test-and-set Primitive", Proc. 1978 International Conference on Parallel Processing, 79-87.

[FR] N. Francez and M. Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme", Proc. 21st Symposium on the Foundations of Computer Science (1980), 373-379.

[LPS] D. Lehmann, A. Pnueli and J. Stavi, "Impartiality, Justice, Fairness: The Ethics of Concurrent Termination", Proc. 8th International Colloquium on Automata, Languages and Programming, Haifa, Israel (1981).

[LR] D. Lehmann and M. O. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem", Proc. 8th Symposium on the Principles of Programming Languages, Williamsburg (1981), 133-138.

[Ra1] M. O. Rabin, "N Process Synchronization by a 4 log N - Valued Shared Variable", Proc. 21st Symposium on the Foundation of Computer Science (1980), 407-410.

[Ra2] M. O. Rabin, "The Choice Coordination Problem", Mem. No.UCB/ERL M80/38, Electronics Research Lab. University of California at Berkeley, August 1981.

[RS] J. Reif and P. Spirakis, "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time", Proc. 13th ACM Symposium on Theory of Computing (1981), 133-145.

[IR] A. Itai and M. Rodeh, "The Lord of the Ring, or Probabilistic Methods for Breaking Symmetry in Distributive Networks", Tech. Rep. RJ 3110, IBM, San Jose 1981.