Reprinted from INFORMATION AND COMPUTATION
All Rights Reserved by Academic Press, New York and London

Vol. 81, No. 1, April 1989
Printed in Belgium

Reasoning about Networks with Many Identical Finite State Processes*

M. C. BROWNE, E. M. CLARKE, AND O. GRUMBERG

Carnegie Mellon University, Pittsburgh, Pennsylvania

1. INTRODUCTION

Consider a distributed mutual exclusion algorithm for processes arranged in a ring network in which mutual exclusion is guaranteed by means of a token that is passed around the ring (Dijkstra 1985, Kurshan 1985, Martin 1985). How can we determine that such a system of processes is correct? Our first attempt might be to consider a reduced system with one or two processes. If we can show that the reduced system is correct and if the individual processes are really identical, then we are tempted to conclude that the entire system will be correct. In fact, this type of informal argument is used quite frequently by designers in constructing systems that contain large numbers of identical processing elements. Of course, it is easy to contrive an example in which some pathological behavior only occurs when, say, 100 processes are connected together. By examining a system with only one or two processes it might even be quite difficult to determine that this behavior is possible. Nevertheless, one has the feeling that in many cases this kind of intuitive reasoning does lead to correct results. The question that we address in this paper is whether it is possible to provide a solid theoretical basis that will prevent fallacious conclusions in arguments of this type.

In addition to providing a firm basis for a common type of informal reasoning, our results are crucial for the success of automatic verification methods that involve *temporal logic model checking* (Clarke, Emerson, and Sistla, 1986; Lichtenstein and Pnueli, 1985; Quielle and Sifakis, 1981; Sistla and Clarke, 1986). These techniques check that a finite-state concurrent system satisfies a temporal logic formula by searching all possible paths in the global state graph determined by the concurrent system. They have been used successfully to find subtle errors in tricky self-timed circuits —errors that were apparently unknown to the designers of the circuits

* This research was partially supported by NSF Grant MCS-82-16706. The third author, O. Grumberg, was on leave from Technion, Haifa, during this period and was partially supported by a Weizmann postdoctoral fellowship.

0890-5401/89 $3.00

Browne *et al.*, 1986; Dill and Clarke, 1986). Although model checking is linear in the size of the global state graph, the number of states in the graph may be exponential in the number of processes. We call this problem the *state explosion phenomenon.* By using the results of this paper, model checking may become feasible for networks with large numbers of identical processes, thus extending the usefulness of this verification method considerably.

The logic that we use for specification is based on computation trees and is called *indexed CTL**, or ICTL*. It includes all of CTL* (Clarke *et al.*, 1986; Emerson and Halpern, 1983) with the exception of the nexttime operator and can, therefore, handle both linear and branching time properties with equal facility. Typical operators include **AG** f, which will hold in a state provided that f holds globally along all possible computation paths starting from that state, and **AF** f, which will hold in a state provided that f eventually holds along all computation paths. In addition, our logic permits formulas of the form $\bigwedge_i f(i)$ and $\bigvee_i f(i)$, where $f(i)$ is a formula of our logic. The subformula $f(i)$ is called a *generic* formula; all of the atomic propositions that appear within it must be subscripted by i. A formula of our logic is said to be *closed* if all indexed propositions are within the scope of either a $\bigwedge_i$ or $\bigvee_i$.

A *model* for our logic is a labeled state transition graph or *Kripke structure* that represents the possible global state transitions of some finite-state concurrent system. For a family of N identical processes this state graph may be obtained as a composition of the state graphs of the individual processes. Instances of the same atomic proposition in different processes are distinguished by using the number of the process as a subscript; thus, A_5 represents the instance of atomic proposition A associated with process 5.

Since a closed formula of our logic cannot contain any atomic propositions with constant index values, it is impossible to refer to a specific process by writing such a formula. Hence, changing the number of processes in a family of identical processes should not affect the truth of a formula in our logic. We make this intuitive idea precise by introducing a new notion of *bisimulation* (Milner, 1979) between two Kripke structures with the same set of indexed propositions but different sets of index values. We then show that if two structures correspond in this manner, a closed formula of indexed CTL* will be true in the initial state of one if and only if it is true in the initial state of the other.

We illustrate these ideas by considering a distributed mutual exclusion algorithm like the one mentioned above. We assume that the atomic proposition c_i is true when the ith process is in its critical region, and that the atomic proposition d_i is true when the ith process is delayed waiting to enter its critical region. A typical requirement for such a system is that a process waiting to enter its critical region will eventually enter the critical region. This condition is easily expressed in our logic by the formula

$$\bigwedge_i \mathbf{AG}(d_i \Rightarrow \mathbf{AF}c_i).$$

In this case, to establish the bisimulation between networks with different numbers of processes it is sufficient to prove that some simple safety properties hold regardless of the size of the network. By using our results it is then possible to show that exactly the same formulas of our logic hold in the network with 1000 processes, as hold in the network with two processes! Thus, we can use the temporal logic model checking algorithm to verify automatically that the above formula holds in the network of size two and conclude that it also holds in the network of size 1000. Although this example is quite simple, it should suggest many potential applications for the results of our paper.

Brookes and Rounds (1983), Hennessy and Milner (1980), and Graf and Sifakis (1985) have all investigated the relationship between temporal logic and various notions of bisimulation among concurrent programs. However, none of the logics in their papers have operators that permit assertions about large numbers of similar processes; consequently, their results are not directly useful in solving the problem that we address in this paper. Kurshan (1985) has studied the state explosion problem in the context of an automatic protocol verification system being developed at Bell Labs. In his system, protocols are verified by showing inclusion between two finite-state machines, one representing the protocol under study and one representing its specification. The state explosion problem is handled by using a homomorphism to collapse a large state machine into a much smaller one while preserving those properties that are important for verification. Since Kurshan does not use temporal logic formulas for specification, he has no analog of our indexed formulas or of our correspondence theorem. In Reif and Sistla (1985) a logic is described that has spatial as well as temporal operators. The spatial operators can range over the processes in a concurrent program and express properties similar to those expressed by our indexed formulas. However, they do not provide a way of collapsing large machines into smaller ones, and even the propositional version of their logic is undecidable. Wolper (1986) also considers a similar logic for reasoning about programs that are data-independent; however, his indexed variables range over data elements, while ours range over processes. Also, there is no notion of correspondence between structures in his work. Some limitations on the type of reasoning that we propose are discussed in Apt and Kozen (1986).

Our paper is organized as follows: In Section 2 we introduce the basic temporal logic CTL*. In Section 3 we state the notion of correspondence or bisimulation that we use between two finite-state machines. We also prove that this notion of bisimulation preserves the truth of CTL* formulas. In Section 4 we extend CTL* to include formulas of the form $\bigwedge_i f(i)$ and $\bigvee_i f(i)$ as explained above. We also extend our notion of correspondence and show that corresponding structures satisfy the same indexed CTL* formulas. Section 5 illustrates how the ideas in this paper can be applied to a concrete example, the distributed mutual exclusion algorithm discussed earlier. The paper ends in Section 6 with some suggestions for possible extensions.

2. The Logic CTL*

There are two types of formulas in CTL*: *state formulas* (which are true in a specific state) and *path formulas* (which are true along a specific path). Let AP be the set of atomic proposition names. A state formula is either:

- A, if $A \in AP$.
- If f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas.
- If f is a path formula, then $\mathbf{E}(f)$ is a state formula.

A path formula is either:

- A state formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, and $f\mathbf{U}g$ are path formulas.

Unless otherwise stated, we will refer to the set of state formulas generated by the above rules as CTL*.

We define the semantics of CTL* with respect to a structure $M = \langle S, R, L, s_0 \rangle$, where

- S is a set of states.
- $R \subseteq S \times S$ is the transition relation, which must be total. We write $s_1 \rightarrow s_2$ to indicate that $(s_1, s_2) \in R$.
- $L: S \rightarrow 2^{AP}$ is the proposition labeling.
- s_0 is the initial state.

We define a *path in M* to be a sequence of states, $\pi = s_0, s_1, \ldots$ such that for every $i \geqslant 0$, $s_i \rightarrow s_{i+1}$. π^i will denote the *suffix* of π starting at s_i. We will sometimes refer to a *prefix* of a path as a path as well.

We use the standard notation to indicate that a state formula f holds in a structure: $M, s \models f$ means that f holds at state s in structure M.

Similarly, if f is a path formula, $M, \pi \models f$ means that f holds along path π in structure M. The relation $\models$ is defined inductively as follows (assuming that f_1 and f_2 are state formulas and g_1 and g_2 are path formulas):

1. $s \models A \Leftrightarrow A \in L(s)$.
2. $s \models \neg f_1 \Leftrightarrow s \not\models f_1$.
3. $s \models f_1 \vee f_2 \Leftrightarrow s \models f_1$ or $s \models f_2$.
4. $s \models \mathbf{E}(g_1) \Leftrightarrow$ there exists a path π starting with s such that $\pi \models g_1$.
5. $\pi \models f_1 \Leftrightarrow s \models f_1$, where s is the first state of π.
6. $\pi \models \neg g_1 \Leftrightarrow \pi \not\models g_1$.
7. $\pi \models g_1 \vee g_2 \Leftrightarrow \pi \models g_1$ or $\pi \models g_2$.
8. $\pi \models g_1 \mathbf{U} g_2 \Leftrightarrow$ there exists $k \geqslant 0$ such that $\pi^k \models g_2$ and for all $0 \leqslant j < k$, $\pi^j \models g_1$.

We will also use the following abbreviations in writing CTL* formulas:

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $\mathbf{F}f \equiv \mathit{true}\ \mathbf{U}f$
- $\mathbf{A}(f) \equiv \neg \mathbf{E}(\neg f)$
- $\mathbf{G}f \equiv \neg \mathbf{F} \neg f$.

We have omitted the nexttime operator, since it can be used to count the number of processes. For example, consider a ring of processes that pass around a token. If t_1 is true when process 1 has the token, then using the nexttime operator $\mathbf{X}$,

$$\mathbf{AG}(t_1 \Rightarrow (\mathbf{XXX}t_1))$$

says that whenever process 1 gets the token it will receive it again in exactly three steps. This is only true if the ring has exactly three processes.

3. Correspondence of Structures

We want to be able to define a correspondence (or bisimulation) between two structures, $M = \langle S, R, L, s_0 \rangle$ and $M' = \langle S', R', L', s'_0 \rangle$ such that if the structures correspond, then one structure satisfies a CTL* formula if and only if the other satisfies it as well. There may be a portion of a path along which several consecutive states are all labeled by the same set of propositions. We will call such a sequence of states a *block*. Since CTL* has no nexttime operator, it is impossible to differentiate between a single state and a block with the same labeling as the state. However, when we correspond a state with a block, we must insure that the block is finite. Therefore, we define a *finite* correspondence relation, $E \subseteq S \times S' \times \mathbf{N}$ which is total for both S and S'. Intuitively, (s, s', k) is in E if state s behaves like state s' and k is an upper bound on the size of the block that will correspond to s' (or s). We will call k the *degree of the correspondence*.

We will write $sE^k s'$ to denote $(s, s', k) \in E$. Also, we will say that two structures, M and M', *correspond* if there is a correspondence relation E between the two structures. Formally, E is a correspondence relation if the following conditions are satisfied:

1. $s_0 E^k s'_0$ for some $k \in \mathbb{N}$. (The initial states should behave similarly.)
2. For every $s \in S$ and $s' \in S'$ such that $sE^k s'$:
   a. For every $A \in AP$, $s \models A \Leftrightarrow s' \models A$. (The proposition labelings are the same.)
   b. $\exists s'_1 [s' \rightarrow s'_1 \wedge sE^v s'_1] \vee \forall s_1 [s \rightarrow s_1 \Rightarrow (s_1 E^v s' \vee \exists s'_1 [s' \rightarrow s'_1 \wedge s_1 E^w s'_1])]$, where $0 \leqslant v < k$ and $w \geqslant 0$.
   c. $\exists s_1 [s \rightarrow s_1 \wedge s_1 E^v s'] \vee \forall s'_1 [s' \rightarrow s'_1 \Rightarrow (sE^v s'_1 \vee \exists s_1 [s \rightarrow s_1 \wedge s_1 E^w s'_1])]$, where $0 \leqslant v < k$ and $w \geqslant 0$.

We will write sEs' to indicate that there exists a k such that $(s, s', k) \in E$. Furthermore, if B and B' are sequences of states, we will write BEB' to indicate that every state in B corresponds to every state in B'.

We will say that two states *exactly match* if for every successor of one state, there is a corresponding successor of the other and vice versa. The above definition ensures an exact match between two states if they correspond with degree 0. If two corresponding states do not exactly match, then the degree of the correspondence sets an upper bound on the total number of transitions that can be made from each state until an exact match is reached. It is easy to prove that the minimal degree of correspondence is equal to the minimal number of transitions until an exact match is reached. Since we never have to make a transition to the same state twice, the minimal number of transitions from each state must be bounded by the

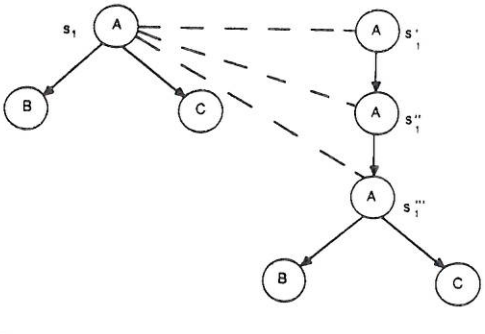

FIG. 3.1. An illustration of corresponding structures.

number of states in the machine. Therefore, the minimal degree of correspondence is bounded by the number of states in the machine as well.

For example in Fig. 3.1, state s_1 exactly matches state s_1''', so these states can correspond with degree 0. State s_1' can reach an exact match with s_1 within 2 transitions, so these two states can correspond with degree 2. Note that the definition of a correspondence relation is not constructive. The above definition can be used to determine if a given relation E is a correspondence relation. However, in its present form the definition cannot be used as the basis for an algorithm to determine if two structures correspond. An algorithm for determining correspondence between structures can be found in (Browne *et al.*, 1987).

We use this intuition to prove the following lemma:

LEMMA 1. *Let M and M' be two structures that correspond. Then, for every $(s, s') \in E$ and for every path π in M that starts in s, there is a path π' in M' that starts in s', a partition of π $(B_1 B_2, \ldots)$, and a partition of $\pi'(B_1' B_2', \ldots)$ such that for all j, $B_j E B_j'$ and both $|B_j|$ and $|B_j'|$ are at least* 1 *and at most $|S| + |S'|$.*

Moreover, for every path π' in M', there is a path π in M and partitions of both paths that satisfy similar conditions.

Proof. First, we will prove this for finite paths by induction on the length of π. First, note that if we consider M and M' to be one structure, it is easy to see that the minimal degree of correspondence between any two states must be bounded by $|S| + |S'|$.

Base. π is of length 1, so $\pi = s$. Let $B_1 = \langle s \rangle$, $\pi' = s'$, and $B_1' = \langle s' \rangle$.

Induction. Let $\pi = s_1 s_2, \ldots, s_n$. By the inductive hypothesis, there is a partition of π, $B_1 B_2, \ldots, B_l$, a path π' in M', and a partition of π', $B_1' B_2', \ldots, B_l'$ such that $B_j E B_j'$ for $1 \leqslant j \leqslant l$. Now we want to show that if we lengthen π by adding some s_{n+1} such that $s_n \rightarrow s_{n+1}$, the lemma still holds.

Since s_n is the last state of π, it must be in the last block B_l, so there must be a k such that $s_n E^k \, \mathbf{last}(B_l')$. We will prove by induction on k that it is possible to extend π' as required.

The basis for the second induction is $s_n E^0 \, \mathbf{last}(B_l')$. By the definition of E^0, there exists a s_1' such that $\mathbf{last}(B_l') \rightarrow s_1' \wedge s_{n+1} E^w s_1'$ for some $w \geqslant 0$. We can extend the partitions of π and π' by defining $B_{l+1} = \langle s_{n+1} \rangle$ and $B_{l+1}' = \langle s_1' \rangle$. Therefore, the basis case is true.

For the inductive step, the definition of E has three cases:

1. $\exists s_1' [\mathbf{last}(B_l) \rightarrow s_1' \wedge s_{n+1} E^w s_1']$ for some $w \geqslant 0$. This case is the same as the base case.

2. $\exists s_1' [\mathbf{last}(B_l') \rightarrow s_1' \wedge s_n E^v s_1']$ for some $0 \leqslant v < k$, but not case 1. If $|B_l| \neq 1$, we can remove the last state, s_n from B_l. Let $\bar{B}_l$ be B_l with s_n

removed, $B_{l+1} = \langle s_n \rangle$, and $B'_{l+1} = \langle s'_1 \rangle$. On the other hand, if $|B_l| = 1$, we can simply add s'_1 to B'_l. In both cases, since the degree of correspondence between s_n and s'_1 is less than k, by the inductive hypothesis, we can extend π' appropriately. Furthermore, since the minimal degree of correspondence between **first**(B_l) and **first**(B'_l) must be bounded by $|S| + |S'|$ and the degree of correspondence decreases as we add states to B'_l, we will put at most $|S| + |S'|$ states into B'_l in this step.

3. $s_{n+1} E^v$ **last**(B'_l) for some $0 \leqslant v < k$, but not case 1. To begin with, if $|B'_l| \neq 1$, we can remove the last element of B'_l and put it into a new block of the partition. Let $\bar{B}'_l$ be B'_l without the last element, $B'_{l+1} = \langle \textbf{last}(B'_l) \rangle$, and $B_{l+1} = \langle s_{n+1} \rangle$. These partitions satisfy the lemma. On the other hand, if $|B'_l| = 1$, we can simply add s_{n+1} to B_l. Furthermore, since the minimal degree of correspondence between **first**(B_l) and **first**(B'_l) must be bounded by $|S| + |S'|$ and the degree of correspondence decreases as we add states to B_l, we will put at most $|S| + |S'|$ states into B_l in this step. Therefore, the lemma holds for this case.

Now that we have proven the lemma for finite paths, we will show that if an infinite path does not have a corresponding infinite path, then there must be a finite prefix of the path that does not have a corresponding path. The argument uses *Konig's lemma.* In order to apply this lemma, we construct a tree in which each node is labeled with a state from S'. π' will be a path through the tree if and only if π' is a path through M' and there is a prefix of π (perhaps consisting of the entire path) and partitions of both paths that satisfy the conditions of the lemma. Note that this tree is finitely branching since M' has only a finite number of states. Furthermore, there can be no infinite path through the tree. If there were, the correspondence to a prefix of π must consist of an infinite number of blocks, since each block must be finite, so the prefix of π must be infinite as well. The only infinite prefix of π is the entire path, which contradicts our assumption. By Konig's lemma, the tree must have an finite number of nodes, and therefore a finite height m. Now consider the prefix of π of length $m \times (|S| + |S'|) + 1$. Since the size of each block is bounded by $|S| + |S'|$, any corresponding path in M' must have at least $m + 1$ blocks. But since the longest path through the tree has only m states, there can be at most m blocks. Therefore, we conclude that this finite prefix has no corresponding path.

Since we have already proven the lemma for finite paths, we can conclude that it holds for infinite paths as well.

Given π' in M', we can use the same argument to show the existence of π in M and the corresponding partitions. Therefore, the lemma holds.

We now prove the *CTL* correspondence theorem.*

THEOREM 2. *Let M_1 and M_2 be two structures that correspond. Then for all $h \in CTL^*$, $M_1, s_0^1 \models h \Leftrightarrow M_2, s_0^2 \models h$.*

This theorem is a consequence of the following lemma:

LEMMA 3. *Let M_1 and M_2 be two structures that correspond. Let h be either a state formula or a path formula. Let π be a path in M_1 starting with s and π' be a path in M_2 starting with s'. If there is a partition of $\pi(B_1 B_2, ...)$ and a partition of $\pi'(B'_1 B'_2, ...)$ such that all of the blocks are finite and $B_j \mathrel{E} B'_j$ for all j, then*

$s \models h \Leftrightarrow s' \models h$, if h is a state formula, and

$\pi \models h \Leftrightarrow \pi' \models h$, if h is a path formula.

Proof. Since $s \in B_1$ and $s' \in B'_1$, sEs'. We will now prove the lemma by induction on the structure of h.

Base. $h = A$. By the definition of E, $s \models A \Leftrightarrow s' \models A$.

Induction. There are several cases:

1. $h = \neg h_1$, a state formula,

$$\begin{aligned} s \models h &\Leftrightarrow s \not\models h_1 \\ &\Leftrightarrow s' \not\models h_1 \qquad \text{(induction hypothesis)} \\ &\Leftrightarrow s' \models h. \end{aligned}$$

The same reasoning holds if h is a path formula.

2. $h = h_1 \vee h_2$, a state formula. Without loss of generality,

$$\begin{aligned} s \models h &\Leftrightarrow s \models h_1 \text{ or } s \models h_2 \\ &\Rightarrow s \models h_1 \\ &\Leftrightarrow s' \models h_1 \qquad \text{(induction hypothesis)} \\ &\Rightarrow s' \models h. \end{aligned}$$

The argument is the same in the other direction. We can also use this argument if h is a path formula.

3. $h = \mathbf{E}(h_1)$, a state formula. Suppose that $s \models h$. Then there is a path, $\pi_1 = ss_1s_2,...$ starting with s such that $\pi_1 \models h_1$. By Lemma 1, there is an partition of this path, $B_1 B_2,...,$ and a path π'_1 in M_2 with a partition, $B'_1 B'_2, ...$ such that the blocks of both partitions are finite and $B_j E B'_j$ for all $j \geqslant 1$. So by the induction hypothesis, $\pi_1 \models h_1 \Leftrightarrow \pi'_1 \models h_1 \Leftrightarrow \pi'_1 \models h_1$. Therefore, $s \models \mathbf{E}(h_1) \Rightarrow s' \models \mathbf{E}(h_1)$. We can use the same argument in the other direction, so the lemma holds.

4. $h = h_1$, where h is a path formula and h_1` is a state formula. Although the lengths of h and h_1 are the same, we can imagine that $h = \mathbf{path}(h_1)$, where **path** is an operator which converts a state formula into a path formula. Therefore, we are simplifying h by dropping this **path** operator. So now,

$$\begin{aligned} \pi \models h &\Leftrightarrow s \models h_1 \\ &\Leftrightarrow s' \models h_1 \qquad \text{(induction hypothesis)} \\ &\Rightarrow \pi' \models h. \end{aligned}$$

The reverse direction is similar.

5. $h = h_1 \mathbf{U} h_2$, a path formula. Suppose that $\pi \models h_1 \mathbf{U} h_2$. By the definition of the until operator, there is a k such that $\pi^k \models h_2$ and for all $0 \leqslant j < k$, $\pi^j \models h_1$. Suppose that s_k is in block B_l. Then, $\bar{B}_l B_{l+1}, ...,$ where $\bar{B}_l$ is the part of B_l starting with s_k, is a partition of π_k. So $B'_l B'_{l+1}, ...$ is the partition of a path in M_2 such that $B_j E B'_j$ is true for all $j \geqslant l$. Therefore, by the induction hypothesis,

$$B'_l B'_{l+1} \cdots \models h_2.$$

Now, any state s'_m before $\mathbf{first}(B'_l)$ on the path π' is in some block B'_j, $j < l$. If $\bar{B}'_j$ is the part of B'_j starting with s'_m, then $\bar{B}'_j B'_{j+1},...$ is a partition of π'^m. Also, $B_j B_{j+1}, ...$ is a partition of a suffix of π such that $B_n E B'_n$ is true for all $n \geqslant j$. Since we know $j < l$, we know that this path starts with a state before s_k, so $B_j B_{j+1},... \models h_1$. Therefore, by the induction hypothesis, $\pi'^m \models h_1$ for any m before $\mathbf{first}(B'_l)$. Therefore $\pi' \models h$.

We can use the same argument in the other direction.

4. Applying CTL* to Networks of Processes

In order to reason about networks of identical processes, we need to be able to distinguish between the atomic propositions of the different processes. Therefore, we introduce the notion of *indexed atomic propositions* such that A_i is the value of proposition A in process i. Let IP be a set of proposition names which will be indexed by a set of index variables, IV, and let AP be a set of atomic propositions as before. The logic *indexed CTL** is an extension of CTL*, where

- A_i is a state formula if $A \in IP$ and $i \in IV$.
- If f is a state formula that has exactly one free index variable i, then $\bigvee_i f$ is a state formula. (We will write $f(i)$ to indicate that f has a free index variable i.)

Indexed CTL* is the set of *closed* state formulas generated by these rules and the rules in Section 2.

We define the semantics of indexed CTL* with respect to a structure $M = \langle AP, IP, I, S, R, L, s_0 \rangle$, where

- AP is the set of atomic formulas.
- IP is the set of atomic formulas indexed by values from I.
- I is the set of index values (a subset of $\mathbf{N}$).
- S is a set of states.
- $R \subseteq S \times S$ is the transition relation.
- $L: S \to 2^{AP \cup (IP \times I)}$ is the proposition labeling. We will write A_i instead of (A, i).
- s_0 is the initial state.

We extend the relation $\models$ to deal with indexed CTL* fromulas as well:

1. $s \models A_c \Leftrightarrow A_c \in L(s)$.
2. $s \models \bigvee_i f_1(i) \Leftrightarrow s \models f_1(c)$.

We will use $\bigwedge_i f(i)$ as an abbreviation for $\neg \bigvee_i \neg f(i)$.

Even without the nexttime operator, this logic is too powerful; by nesting the operators $\bigwedge_i$ and $\bigvee_i$ it might still be possible to count the number of processes in a concurrent system. Suppose we take as our Kripke structure the global state graph for the concurrent program in Fig. 4.1. The following formula sets a lower bound on the number of processes:

$$\bigvee_i \left(A_i \wedge \mathbf{EF}\left(B_i \wedge \bigvee_j \left(A_j \wedge \mathbf{EF}\left(B_j \wedge \bigvee_k (A_k, \ldots)\right)\right)\right)\right).$$

Once B_i becomes true, it remains true. Therefore, if $\bigvee_k A_k$ is true, we know that this k is different from all of the preceding indices mentioned in the formula. For this reason, we will use a restricted form of ICTL*. The additional restrictions are:

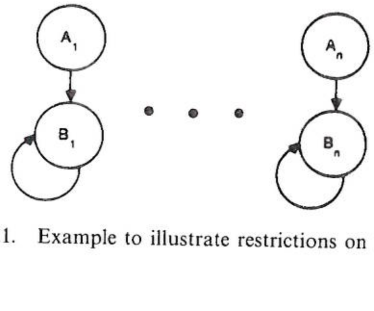

FIG. 4.1. Example to illustrate restrictions on ICTL*.

- $\bigvee_i f$ is a permissible state formula only if f does not contain any $\bigvee_j$ operators.

- $g_1 \mathbf{U} g_2$ is a permissible path formula only if neither g_1 nor g_2 contains any $\bigvee_j$ operators.

In practice, many of the most interesting properties of networks of identical processes can be expressed in the restricted logic. One important property that cannot be expressed is that an indexed proposition holds for *exactly one* index value, since this involves nesting of $\bigwedge_i$ operators. Nevertheless, we can handle such a property within the framework that we have developed by means of a slight extension to the language and its semantics. We add a special non-indexed atomic formula, $\oplus_i P_i$ to AP for every P in IP. The proposition labeling is then extended as follows: $\oplus_i P_i \in L(s)$ if and only if there is exactly one $c \in I$ such that $c \in I$ such that $P_c \in L(s)$. In the remainder of the paper, we will refer to the restricted logic with this extension as ICTL* unless otherwise stated.

We can use the notion of correspondence defined in Section 3 to define an *indexed correspondence* between two structures M and M' with sets of index values I and I', respectively. Since the restrictions to ICTL* do not permit the use of two different indices with an "until" operator, it is impossible to refer to the behavior of two different processes along a specific path. Thus, the notion of indexed correspondence between structures only needs to refer to one index value from each structure at a time. Because of this, we will define a set of correspondence relations, E'_{ii}, that relate the behavior of an index i in I to the behavior of an index i' in I'.

Let M be a structure and i be an index value from I. The *reduction of M to i* (denoted by $M|_i$) is a structure identical to M except that the new proposition labeling L_i is defined as

$$L_i(s) = L(s) \cap (AP \cup IP \times \{i\}).$$

In other words, all of the indexed atomic formulas are omitted except those that are indexed by i.

Now, we say that two structures, M and M' with the same set of indexed and nonindexed atomic formulas, *(i, i')-correspond* if and only if $M|_i \, E \, M'|_{i'}$. We will write this as $M \, E_{ii'} \, M'$.

We can prove an analogous result to Lemma 1 for (i, i')-corresponding structures, where the correspondence between states is now an (i, i')-correspondence. Using this result, we can prove the following lemma concerning unquantified formulas:

LEMMA 4. *Let M and M' be two structures that (i, i')-correspond. Let $h(i)$ be an indexed CTL* formula without any $\bigwedge_i$ operators and with one free* *index variable. Let π be a path in M starting with s and π' be a path in M' starting with s'. If there is a partition of $\pi(B_1 B_2, ...)$ and a partition of $\pi'(B'_1 B'_2,...)$ such that all of the blocks are finite and $B_j E_{ii'} B'_j$ for all j then*

$$s \models h(i) \Leftrightarrow s' \models h(i'), \text{ if } h \text{ is a state formula, and}$$
$$\pi \models h(i) \Leftrightarrow \pi' \models h(i'), \text{ if } h \text{ is a path formula.}$$

The proof follows the same lines as the proof of the CTL* correspondence theorem except that there is an extra base case for indexed atomic propositions. By the definition of (i, i')-correspondence, $s \models A_i \Leftrightarrow s' \models A_{i'}$ is immediate.

Using this lemma, we can prove the major result of this paper, the *ICTL* correspondence theorem*:

THEOREM 5. *Let M and M' be two structures and IN be a relation over $I \times I'$ that is total for both I and I'. If for every $(i, i') \in IN$, the two structures (i, i')-correspond, then $M, s_0 \models h \Leftrightarrow M', s'_0 \models h$ for every ICTL* formula h.*

Proof. We prove this theorem by induction on the structure of h. The only interesting case is the base case, when $h = \bigvee_i h_1(i)$. If $s_0 \models \bigvee_i h_1(i)$, then there is some i_0 such that $s_0 \models h_1(i_0)$. Since IN is total, there is an i'_0 such that $(i_0, i'_0) \in IN$. Therefore, since M and M'-correspond, Lemma 4 gives $s'_0 \models h_1(i'_0)$. Therefore, $s'_0 \models \bigvee_i h_1(i)$. The reverse argument is similar. The other base case, $h = A \in AP$, is straightforward.

The proof of the remaining cases ($\neg h_1$ and $h_1 \vee h_2$) are straightforward. Therefore, the ICTL* correspondence theorem is true.

5. Distributed Mutual Exclusion Example

In this section we illustrate how our ideas might be applied to the distributed mutual exclusion example mentioned in the introduction. We assume that r processes are arranged in a ring. Each process P_i is always in one of three states: A *neutral* state (denoted by n_i), a *delay* state (denoted by d_i), or a *critical* state (denoted by c_i). Exactly one process will have the token at any given time; if process i has the token this will be denoted by t_i. The global state graph for the case of two processes is shown in Fig. 5.1. In the case of $r > 2$ processes, there may be more than one delayed process. Whenever this occurs, the process P_i with the token should eventually give the token to the closest neighbor to its left that is in a delay state; we denote the closest neighbor to the left by $\text{cln}(i)$.[1] We next define the state

[1] It is assumed that the token will be transferred through consecutive processes from P_i to $P_{\text{cln}(i)}$, however the exact mechanism of this transfer will not be explicitly represented in our model at this level of abstraction. Thus, the transfer of the token only requires one global transition.

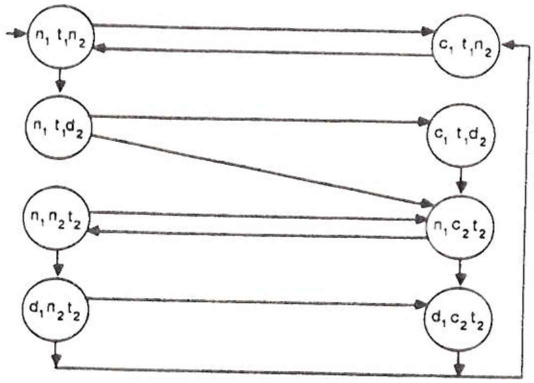

FIG. 5.1. Two-process mutual exclusion example.

transition graph in the case of r processes: $G_r = \langle AP, IP, I_r, S_r, R_r, L_r, s_0^r \rangle$. where

- $AP = \varnothing$
- $IP = \{d, c, n, t\}$
- $I_r = \{1, ..., r\}$
- $S_r = \{s \mid s = \langle D, N, T, C, O \rangle\}$, where each of D, N, T, C, and O is a subset of I_r. We will refer to these subsets as the *parts of state s*. Intuitively, $i \in D$ means that process i is in its delay state. Similarly, $i \in N$ means that P_i is in its neutral state without the token, $i \in T$ means that P_i is in its neutral state with the token, and $i \in C$ means that P_i is in its critical state with the token. Finally, $i \in O$ means that none of the above conditions hold.

- $$\begin{aligned} R_r = \{(s, s_1) \mid s = \langle D, N, T, C, O\rangle \wedge s_1 = \langle D_1, N_1, T_1, C_1, O_1 \rangle \\ \wedge\ [\exists i[i \in N \wedge D_1 = D \cup \{i\} \wedge N_1 = N - \{i\} \wedge T_1 = T \wedge C_1 = C] \\ \vee\ \exists i\, \exists j[i \in D \wedge j \in T \cup C \wedge i = \text{cln}(j) \\ D_1 = D - \{i\} \wedge N_1 = N \cup \{j\} \\ T_1 = T - \{j\} \wedge C_1 = C - \{j\} \cup \{i\}] \\ \vee\ \exists i[i \in T \wedge D_1 = D \wedge N_1 = N \wedge T_1 = T - \{i\} \wedge C_1 = C \cup \{i\}] \\ \vee\ \exists i[i \in C \wedge D = \varnothing \wedge D_1 = D \wedge N_1 = N \\ T_1 = T \cup \{i\} \wedge C_1 = C - \{i\}]]\}. \end{aligned}$$

In the first transition some process moves from its neutral state to its delay state. In the second transition a token is transferred from a process P_j to a process P_i, where $i = \text{cln}(j)$. In the third transition a process with a token moves from its neutral state to its critical state. In the last transition a process with a token moves from its critical state to its neutral state; since no other process wants to token, it remains with the same process.

- $L_r(s) = \{d_i \mid i \in D\} \cup \{n_i \mid i \in N\} \cup \{n_i, t_i \mid i \in T\} \cup \{c_i, t_i \mid i \in C\}$
- $s_0^r = \langle \varnothing, \{2, ..., r\}, \{1\}, \varnothing, \varnothing \rangle$.

Ultimately, we want to establish a correspondence between the mutual exclusion program with r processes and the program with 2 processes. (It is impossible to establish a correspondence between the r process version and the one process since no process can enter its delay state in the one process version.) In order to prove the correctness of the correspondence, we must show that certain simple invariants hold in our mutual exclusion program:

1. D, N, T, and C form a partiton of I_r, i.e., they are disjoint and O is always empty.

2. Once a process has requested the token, it continues to request it until the token is received,

$$\bigwedge_i \mathbf{AG}(d_i \Rightarrow \neg \mathbf{E}[d_i \mathbf{U} \neg d_i \wedge \neg t_i]).$$

3. There is exactly one process with the token at any time, $\mathbf{AG} \oplus_i t_i$.

To establish these invariants, it is sufficient to show that they hold initially in s_0^r and every transition in R_r preserves them. In this case, the proofs are trivial, so we omit them.

The state transition graph given above is not a Kripke structure since some states may not have any transitions (i.e., the state where all processes are delayed and no process has the token). However, if we restrict G_r to be defined over the set of states reachable from s_0^r we do obtain a Kripke structure which we denote by M_r. Since we have shown that every reachable state has a process with the token, this process can always make the transition to and from its critical section; therefore R_r is total.

In order to define the bisimulation between M_2 and M_r, we must first define the relation $IN \subseteq I_2 \times I_r$ that determines the correspondence between index values in the two structures:

$$IN = \{1, 1\} \cup \{(2, i) \mid i \in I_r - \{1\}\}.$$

Next, we must define the correspondence between states $E_{ii'} \subseteq S_2 \times S_r \times \mathbf{N}$ for every $(i, i') \in IN$:

1. Two states, s in M_2 and s' in M_r, (i, i')-correspond if i is in the same part of s as i' is in s' and if $i \in C$ then $D = \varnothing \Leftrightarrow D' = \varnothing$.

2. Let an *i-idle transition* be a transition which does not have any

effect on i, i.e., i belongs to the same part of the state before and after the transition and if $i \in C$ and D is empty, then D remains empty. We define the *rank of s*, $r(s, i)$, to be the maximal number of consecutive i-idle transitions possible from s, if this number is finite. Otherwise, the rank of s is 0. The degree of the correspondence between s and s' is defined to be $r(s, i) + r(s', i')$.

In the Appendix, we show that E satisfies the requirements of a correspondence relation. Once we have established this, we can use the CTL model checking algorithm (Clarke, Emerson, Sistla, 1986) to establish the following properties:

1. A token is transferred only upon request:

$$\neg \bigvee_i \mathbf{EF}(\neg d_i \wedge \neg t_i \wedge \mathbf{E}[\neg d_i \wedge \neg t_i \mathbf{U} t_i]).$$

2. Only the process with a token may get into its critical state:

$$\bigwedge_i \mathbf{AG}(c_i \Rightarrow t_i).$$

3. If a process requests the token, then it will eventually receive it:

$$\bigwedge_i \mathbf{AG}(d_i \Rightarrow \mathbf{A}[d_i \mathbf{U} t_i]).$$

4. Every process that wants to enter its critical state, eventually does:

$$\bigwedge_i \mathbf{AG}(d_i \Rightarrow \mathbf{AF} c_i).$$

6. Directions for Future Research

The notion of bisimulation introduced in Section 4 currently requires some representation for the global states of a product machine. When the individual processes in such a product are more complicated than the ones in the ring network example of Section 5, it may be difficult to find such a representation. Perhaps, an appropriate notion of bisimulation can be found that applies directly to the individual processes rather than to the global state graph. More work clearly needs to be done on this problem. Another problem concerns the restriction on nesting of $\bigwedge_i$'s and $\bigvee_i$'s given in Section 4. We showed how nesting of these operators could be used to count the number of processes in a concurrent program, so some restriction is clearly necessary. We conjecture that with formulas having at most k operators of this type, it is impossible to distinguish between programs that have more than k processes. In other words, if f is a formula with k

levels of $\bigwedge_i$ and $\bigvee_i$ operators and M_n is a Kripke structure obtained as a product of n identical processes, then f will hold in M_n for $n > k$ if and only if f holds in M_k. It is easy to prove this result when the product of the individual processes is a free product, i.e., when there is no synchronization between the individual processes. When the processes are synchronized the conjecture seems much more difficult to prove, however.

APPENDIX: Proof of the Correspondence in Section 5

We assume that the relation E and the rank of a state, $r(s, i)$ are defined as in Section 5. Note that the only case in which the number of consecutive i-idle transitions from s is infinite is when $s \models n_i$. Also note that if s_1 is reachable from s by pursuing i-idle transitions only and if $r(s, i) \neq 0$, then $r(s_1, i) < r(s, i)$.

First, we show how to compute $r(s, i)$. There are a number of cases, depending on which part of the state i is in:

1. $i \in N$. In this case, there are an infinite number of consecutive i-idle transitions starting from s, so $r(s, i) = 0$.

2. $i \in D$. Let process j be the one with the token. There are four sources of i-idle transitions in this case:

   a. Processes that are initially neutral may be come delayed. ($|N|$ transitions.)

   b. The process with the token may enter its critical section. ($|T|$ transitions.)

   c. The token may be transferred to a delayed process between j and i. ($(j-i) \bmod n-1$ transitions.)

   d. The processes that gave up the token in the previous step may become delayed. ($(j-i) \bmod n-1$ transitions.)

Therefore, $r(s, i) = |N| + |T| + 2(j-i) \bmod n-2$.

3. $i \in T$. The only i-idle transitions are neutral processes becoming delayed. So $r(s, i) = |N|$.

4. $i \in C$ and $D = \varnothing$. Since all transitions either move i into a different part of the state or add processes to D, $r(s, i) = 0$.

5. $i \in C$ and $D \neq \varnothing$. The only i-idle transitions are neutral processes becoming delayed. Therefore, $r(s, i) = |N|$.

Now, we must check that E is a correspondence relation.

Clause (1). Because all of the processes are neutral in the initial states of M_2 and M_r, these states correspond for every $(i, i') \in IN$, with a degree $k = r(s_0^1, i) + r(s_0^2, i')$.

Clause (2a). Immediately from the definition of $E_{ii'}$, for every two states s, s' that (i, i')-correspond with any degree, $s \models A_i \Leftrightarrow s' \models A_{i'}$ for every $A \in IP$.

Clause (2b). Assume $sE_{ii'}^k s'$, where $k = r(s, i) + r(s', i')$. There are five cases, one for each of the clauses in the definition of $r(s, i)$. We check the first two cases; the others are similar.

1. $i \in N$ and $i' \in N'$. From above, $r(s, i) = r(s', i') = 0$, so $k = 0$. From s, two kinds of transitions are possible:

   a. Process ii can become delayed in state s_1. Since $i' \in N$, process i' can also become delayed in some state s'_1. These two next states are $E_{ii'}^w$ related, since $i \in D_1$ and $i' \in D'_1$.

   b. Some process can make an i-idle transition to state s_1. In this case, some process in M_r can also make an i'-idle transition to s'_1. Since i and i' are still in the same part, these two next states are $E_{ii'}^0$ related.

Since every transition from s has a corresponding transition from s', Clause (2b) holds in this case.

2. $i \in D$ and $i' \in D'$. There are three cases:

   a. Some process can make an i-idle transition to a state s_1. Since $i \in D$, $s_1 E_{ii'}^v s'$ for $v = r(s_1, i) + r(s', i')$. $r(s, i)$ measures the maximum possible number of i-idle transitions from s. Because an i-idle transition from s has been made, $r(s_1, i) < r(s, i)$ so $v < k$, so Clause (2b) holds.

   b. Process i receives the token from process j and process i' can receive the token from process j'. After these transitions, both i and i' are in C, so the successor states correspond.

   c. Process i receives the token from process j, but process i' cannot receive the token from process j' $(i' \neq \text{cln}(j'))$. Thus, there must be a delayed process between j' and i' which is the closest neighbor of j'. Therefore, there is an i'-idle transition in which this closest neighbor receives the token. The resulting state, s'_1, corresponds to s with degree $v = r(s, i) + r(s'_1, i')$. Since an i'-idle transition from s' has been made, $r(s'_1, i') < r(s', i')$ so $v < k$, so Clause (2b) holds.

Clause (2c). Proven similarly to Clause (2b).
This completes the proof of the bisimulation of M_2 and M_r.

ACKNOWLEDGMENTS

We would like to acknowledge Prasad Sistla, Nissim Francez, and the anonymous referees for their insightful comments on early versions of this paper.

RECEIVED March 7, 1987; ACCEPTED July 5, 1987

REFERENCES

APT, K., AND KOZEN, D. (1986), Limits for automatic verification of finite-state concurrent systems. *Inform. Process. Lett.* **22**, No. 6, 307–309.

BROOKES, S. D., AND ROUNDS, W. C. (1983), Behavioural equivalence relations induced by programming logics, *in* "10th ICALP, 1983," Lect. Notes in Comput. Sci. Vol. 154, Springer-Verlag, New York.

BROWNE, M., CLARKE, E., DILL, D., AND MISHRA, (1986), Automatic verification of sequential circuits using temporal logic, *IEEE Trans. Comput.* **C-35**, No. 12.

BROWNE, M. C., CLARKE, E. M., AND GRUMBERG, O. (1987), Characterizing Kripke structures in temporal logic, *in* "Colloquium on Trees in Algebra and Programming," Pisa, Italy, March; (1988), *Theoret. Comput. Sci.* **59**, 115–131.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. (1986), Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Trans. Programm. Lang. Systems* **8**, No. 2, 244–263.

DIJKSTRA, E. (1985), Invariance and non-determinacy, *in* "Mathematical Logic and Programming Languages" (C.A.R. Hoare and J. C. Shepherdson, Eds.), pp. 157–163, Prentice–Hall, Englewood Cliffs, NJ.

DILL, D. L., AND CLARKE, E. M. (1986), Automatic verification of asynchronous circuits using temporal logic, *IEE-E Proc.* **5**.

EMERSON, E. A., AND HALPERN, J. Y. (1983), "Sometimes" and "not never" revisited: On branching versus linear time, *in* "Proceedings 10th ACM Symp. on Principles of Programming Languages."

GRAF, S., AND SIFAKIS, J. (1985), From synchronization tree logic to acceptance model logic, *in* "Logics of Programs," Lect. Notes in Comput. Sci. Vol. 193, Springer-Verlag, New York/Berlin.

HENNESSY, M. AND MILNER, R. (1980), On observing nondeterminism and concurrency, *in* "7th ICALP," Lect. Notes in Comput. Sci. Vol. 85, Springer-Verlag, New York/Berlin.

KURSHAN, R. P. (1985), Modelling concurrent processes, *in* "Proceedings, Symposia in Applied Mathematics."

LICHTENSTEIN, O., AND PNUELI, A. (1985), Checking that finite state concurrent programs satisfy their linear specification, *in* "Conference Record of the Twelth Annual ACM Symposium on Principles of Programming Languages," New Orleans, LA, January.

MARTIN, A. (1985), The design of a self-timed circuit for distributed mutual exclusion, *in* "Proceedings, Chapel Hill Conf. on VLSI," pp. 247–260.

MILNER, R. (1979), "A Calculus of Communicating Systems," Lect. Notes in Comput. Sci. Vol. 92, Springer-Verlag, New York/Berlin.

QUIELLE, J. P., SIFAKIS, J. (1981), Specification and verification of concurrent systems in CESAR, *in* Proceedings, Fifth International Symposium in Programming.

REIF, J., AND SISTLA, P. (1985), A multiprocess network logic with temporal and spatial modalities, *J. Comput. System Sci.* **30**, No. 1.

SISTLA, A. P., AND CLARKE, E. M. (1986), Complexity of propositional temporal logics, *J. Assoc. Comput. Mach.* **32**, No. 3, 733–749.

WOLPER, P. (1986), Expressing interesting properties of programs in propositional temporal logic, *in* "Thirteenth ACM Symposium on Principles of Programming Languages."

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium