REASONING ABOUT NETWORKS WITH MANY IDENTICAL FINITE-STATE PROCESSES

E.M. Clarke, O. Grumberg, and M.C. Browne

Carnegie-Mellon University
Computer Science Department
Pittsburgh, PA 15213-3890

December 1987

Interim

Approved for Public Release; Distribution is Unlimited
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

CHABIRA M. HOPPER
Project Engineer

RICHARD C. JONES
Ch, Advanced Systems Research Gp
Information Processing Technology Br

FOR THE COMMANDER

EDWARD L. GIATTI
Ch, Information Processing Technology Br
Systems Avionics Div

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFMAJ/AAAT, Wright-Patterson AFB, OH 45433-6510 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.
**Title:** Reasoning About Networks With Many Identical Finite-State Processes

**Personal Authors:**
E. M. Clarke, O. Grumberg, M. C. Browne

**Type of Report:** Interim

**Date of Report:** December 1987

**Abstract:**

Reasoning about networks with many identical finite-state processes involves dealing with the complexity that arises from the large number of states and transitions. This report presents a framework for reasoning about such networks, focusing on techniques that can be applied to verify properties of the system. The approach is based on model checking and involves constructing a symbolic representation of the network to facilitate analysis.

The report discusses the challenges associated with reasoning about networks of this kind and presents methods for addressing these challenges. It includes details on how to encode the networks, how to perform the reasoning, and how to interpret the results. The techniques described are designed to be general enough to be applicable to a wide range of network configurations, making them valuable tools for researchers and practitioners in the field of computer science and related areas.

This work is significant because it provides a systematic approach to a problem that is otherwise computationally intractable. By providing a method to reason about these networks, it opens up new possibilities for understanding and managing complex systems, which can have applications in various domains, including computer security, software verification, and system design.
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 ([6], [10], [12]). 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 ([4], [11], [14], [16]). 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 ([3], [5]). 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* ([4], [7]) with the exception of the next-time operator and can, therefore, handle both linear and branching time properties with equal facility. Typical operators include $\text{AG} f$, which will hold in a state provided that $f$ holds globally along all possible computation paths starting from that state and $\text{AF} f$, 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 $\bigwedge_i$ or $\bigvee_i$.

A model for our logic is a labelled 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, $A_i$ represents the instance of atomic proposition $A$ associated with process $i$.

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 effect the truth of a formula in our logic. We make this intuitive idea precise by introducing a new notion of bisimulation [13] 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 TiT 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 $i$-th process is in its critical region, and that the atomic proposition $d_i$ is true when the $i$-th 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

$$\forall i . GF (d_i \Rightarrow A F c_i).$$

By using our results it is possible to show that exactly the same formulas of our logic hold in a network with 1000 processes as hold in a network with two processes! We can use one of the temporal logic model checking algorithms to automatically check that the above formula holds in networks of size two and conclude that it will also hold in networks of size 1000. Although this example is quite simple, it should suggest many potential applications for the results of our paper.

Brookes and Rounds [2], Hennessy and Milner [9], and Graf and Sifakis [8] 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 [10] 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 analogue of our indexed formulas or of our correspondence theorem. In [15] Reif and Sistla describe a logic 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 also considers a similar logic for reasoning about programs that are data-independent [17]; 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 [1].

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 \( \bigwedge f(i) \) and \( \bigvee f(i) \) 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 \( f \lor g \) are state formulas.
- If \( f \) is a path formula, then \( E(f) \) is a state formula.

A path formula is either:

- A state formula.
- If \( f \) and \( g \) are path formulas, then \( \neg f, f \lor g, \) and \( f \lor g \) are path formulas.

CTL* is the set of state formulas generated by the above rules.

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 \( s \rightarrow s' \) to indicate that \( (s, s') \in R \).
- \( L : S \rightarrow \mathcal{P}(AP) \) 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 \( i \geq 0 \), \( s_i \rightarrow s_{i+1} \). \( \pi' \) will denote the suffix of \( \pi \) starting at \( s_i \).

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 \( \pi \) is a path formula, \( M, \pi \models f \) means that \( f \) holds along path \( \pi \) 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. \( s \models A \iff A \in L(s) \).
2. \( s \models \neg f_i \iff s \not\models f_i \).
3. \( s \models f_1 \lor f_2 \iff s \models f_1 \) or \( s \models f_2 \).
4. \( s \models E(g_1) \iff \) there exists a path \( \pi \) starting with \( s \) such that \( \pi \models g_1 \).
5. \( \pi \models f_i \iff s \) is the first state of \( \pi \) and \( s \models f_i \).
6. \( \pi \models \neg g_i \iff \pi \not\models g_i \).
7. \( \pi \models g_1 \lor g_2 \iff \pi \models g_1 \) or \( \pi \models g_2 \).
8. \( \pi \models g_1 U g_2 \iff \) there exists a \( k \geq 0 \) such that \( \pi^k \models g_2 \), and for all \( 0 \leq j < k \), \( \pi^j \models g_1 \).

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

\( \cdot f \land g \equiv \neg(\neg f \lor \neg g) \) \hspace{2cm} \( \cdot Ff \equiv \text{true} U f \)

\( \cdot M(f) \equiv \neg E(\neg f) \) \hspace{2cm} \( \cdot Gf \equiv \neg F \neg f \).

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 \( t \) is true when process 1 has the token, then using the nexttime operator \( X \),

\[ AG(t) \Rightarrow (XXX t_1) \]

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, \( M_1 \) and \( M_2 \) 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 labelled 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, $E \subseteq S \times S \times \mathbb{N}$ which is total for both $S$ and $S_\ast$. 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 $s E^k s'$ to denote $(s, s', k) \in E$. Also, we will say that two structures, $M_i$ and $M_j$, 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. $s E^k s_0'$ for some $k \in \mathbb{N}$. (The initial states should behave similarly.)

2. For every $s \in S_i$ and $s' \in S_j$ such that $s E^k s'$:
   a. For every $A \in AP, s \models A \Rightarrow s' \models A$. (The proposition labelings are the same.)
   b. $\exists s' [s' \rightarrow s' \land s E^v s'] \forall s [s \rightarrow s \Rightarrow (s \rightarrow s' \land s E^w s')]$
      where $0 \leq v < k$ and $w \geq 0$.
   c. $\exists s [s \rightarrow s \land s E^v s'] \forall s' [s' \rightarrow s' \Rightarrow (s \rightarrow s' \land s E^w s')]$
      where $0 \leq v < k$ and $w \geq 0$.

We will write $s E s'$ 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 $B \models E B'$ 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 insures an exact match between two states if they correspond with degree 0. For example, in Figure 3-1, state $s_1$ exactly matches state $s_1'''$, so these states can correspond with degree 0. If two corresponding states don't exactly match, then the degree of the correspondence sets an upper bound on the number of transitions until an exact match is reached. In the figure, state $s_1'$ can reach an exact match with $s_1$ within 2 transitions, so these two states can correspond with degree 2.

We use this intuition to prove the following lemma:

**Lemma 1:** Let $M_i$ and $M_j$ be two structures that correspond. Then, for every $(s, s') \in E$ and for every path $\pi$ in $M_i$ that starts in $s$, there is a path $\pi'$ in $M_j$ that starts in $s'$, a partition of $\pi$ $(B_1, B_2, \ldots)$, and a partition of $\pi' (B'_1, B'_2, \ldots)$ such that for all $j$, $B_j \models E B'_j$ and either
Figure 3-1: An Illustration of Corresponding Structures

1. \(|B_j| = 1\) and \(B_j'\) is finite, or
2. \(|B_j'| = 1\) and \(B_j\) is finite.

Moreover, for every path \(\pi'\) in \(M_\omega\), there is a path \(\pi\) in \(M_i\) and partitions of both paths that satisfy similar conditions.

Proof: We will prove this by induction on the length of \(\pi\).

Base: \(\pi\) 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 \(\pi, B_1, B_2, \ldots, B_i,\) a path \(\pi'\) in \(M_\omega\), and a partition of \(\pi', B_1' B_2' \ldots B_i'\) such that \(B_j \in E' B_j'\) for \(1 \leq j \leq l\). Now we want to show that if we lengthen \(\pi\) 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 \(\pi\), it must be in the last block \(B_i\), so there must be a \(k\) such that \(s_n E^k \text{last}(B_j')\). We will prove by induction on \(k\) that it is possible to extend \(\pi'\) as required.

The basis for the second induction is \(s_n E^0 \text{last}(B_j')\). By the definition of \(E^0\), there exists a \(s'_1\) such that \(\text{last}(B_j') \rightarrow s'_1 \land s_{n+1} E^w s'_1\) for some \(w \geq 0\). We can extend the partitions of \(\pi\) and \(\pi'\) by defining \(B_{i+1} = \langle s_{n+1} \rangle\) and \(B'_{i+1} = \langle s'_1 \rangle\). Therefore, the basis case is true.
For the inductive step, the definition of $F$ has three cases:

1. $\exists s'_n \text{ [last}(B_i') \rightarrow s'_n \land s_{n+1} \in s' \} \text{ for some } w \geq 0.$

   This case is the same as the base case.

2. $\exists s'_n \text{ [last}(B_i') \rightarrow s'_n \land s_n \in s' \} \text{ for some } 0 \leq v < k.$

   If $|B_i'| \neq 1$, we can remove the last state, $s_n$ from $B_i'$. Let $\bar{B}_i$ be $B_i'$ with $s_n$ removed. $B_{i+1}' = \langle s_n \rangle$ and $B_{i+1}' = \langle s'_n \rangle$. On the other hand, if $|B_i'| = 1$, we can simply add $s'_n$ to $B_i'$. In both cases, since the degree of correspondence between $s_n$ and $s'_n$ is less than $k$, by the inductive hypothesis, we can extend $\pi'$ appropriately.

3. $s_{n+1} \in \text{ last}(B_i') \text{ for some } 0 \leq v < k.$

   To begin with, if $|B_i'| \neq 1$, we can remove the last element of $B_i'$ and put it into a new block of the partition. Let $\bar{B}_i'$ be $B_i'$ without the last element, $B_{i+1}' = \langle \text{last}(B_i') \rangle$, and $B_{i+1}' = \langle s_{n+1} \rangle$. These partitions satisfy the lemma.

   On the other hand, if $|B_i'| = 1$, we can simply add $s_{n+1}$ to $B_i$. Therefore, the lemma holds for this case.

It is also necessary to show that all of the blocks in this construction are finite. This problem may arise in the second and the third case, where we might add an infinite number of states to $B_i'$ (or $B_i$). However, since the degree of correspondence between the states in $B_i'$ ($B_i$) and the state in $B_i'$ ($B_i$) is decreasing and cannot be less than zero, these constructions will only apply a finite number of times. Hence, only a finite number of states will be added to the last block, so it must be finite.

Given $\pi'$ in $M_1$, we can use the same argument to show the existence of $\pi$ in $M_2$ and the corresponding partitions. Therefore, the lemma holds. $\Box$

We now prove the $CTL^*$ correspondence theorem:

**Theorem 2:** Let $M_1$ and $M_2$ be two structures that correspond. Then for all $h \in CTL^*$,

$M_1, s_0 \models h \Rightarrow M_2, s_0' \models h.$

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 $\pi$ be a path in $M_1$ starting with $s$ and $\pi'$ be a path in $M_2$ starting with $s'$. If there is a partition of $\pi$ ($B_1, B_2, \ldots$) and a partition of $\pi'$ ($B'_1, B'_2, \ldots$) such that all of the blocks are finite and $B_j \subseteq B_j'$ for all $j$, then

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

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

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

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

Induction: There are several cases.

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

$s \models h \Rightarrow s \not\models h$

$\Rightarrow s' \not\models h_1$ (induction hypothesis)

$\Rightarrow s' \models h$

The same reasoning holds if $h$ is a path formula.

2. $h = h_1 \lor h_2$, a state formula.

Without loss of generality,

$s \models h \Rightarrow s \models h_1$ or $s \models h_2$

$\Rightarrow s \models h_1$

$\Rightarrow s' \models h_1$ (induction hypothesis)

$\Rightarrow s' \models h$

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

3. $h = E (h_1)$, a state formula.

Suppose that $s \models h$. Then there is a path, $\pi_1 = s_1 s_2 \ldots$ starting with $s$ such that $s_1 \models h_1$. By Lemma 1, there is an partition of this path, $B_1 B_2 \ldots$, and a path $\pi'_1$ in $M_2$ with a partition, $B'_1 B'_2 \ldots$ such that the blocks of both partitions are finite and $B_j \subseteq B'_j$ for all $j \geq 1$. So by the induction hypothesis, $s_1 \models h_1 \Rightarrow \pi'_1 \models h_1$. Therefore, $s \models E (h_1) \Rightarrow s' \models E (h_1)$. 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 \( h' \) are the same, we can imagine that \( h = \text{path}(h) \), 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:

\[
\begin{align*}
\pi & \equiv h' \\
\pi' & \equiv h, \quad \text{(induction hypothesis)}
\end{align*}
\]

The reverse direction is similar.

5. \( h = h \cup h' \), a path formula.

Suppose that \( \pi \equiv h \cup h' \). By the definition of the until operator, there is a \( k \) such that \( \pi^k \equiv h' \) and for all \( 0 \leq j < k \), \( \pi'^j \equiv h' \). Suppose that \( s_k \) is in block \( B_i \). Then, \( B_i, B_{i+1}, \ldots, \) where \( B_i \) is the part of \( B \) starting with \( s_k \), is a partition of \( \pi^k \). So \( B_i', B_{i+1}', \ldots \) is the partition of a path in \( M \), such that \( B_i', B_{i+1}', \ldots \) is true for all \( j \geq k \). Therefore, by the induction hypothesis,

\[
B_i', B_{i+1}', \ldots \Rightarrow h.
\]

Now, any state \( s_m \) before \( \text{first}(B_i') \) on the path \( \pi' \) is in some block \( B_{j+1}' \). If \( B_{j+1}' \) is the part of \( B_j' \) starting with \( s_{j+1} \), then \( B_j', B_{j+1}', \ldots \) is a partition of \( \pi^m \). Also, \( B_j', B_{j+1}', \ldots \) is a partition of a suffix of \( \pi \) such that \( B_j, B_{j+1}, \ldots \) is true for all \( n \geq j \). Since we know \( j < k \), we know that this path starts with a state before \( s_k \), so \( B_j, B_{j+1}, \ldots \Rightarrow h \). Therefore, by the induction hypothesis,

\[
\pi^m \Rightarrow h
\]

for any \( m \) before \( \text{first}(B_i') \). Therefore \( \pi' \equiv h \).

We can use the same argument in the other direction. \( \square \)

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 $M = (\mathcal{F}, \mathcal{P}, L, R, I, S)$, where:
- $\mathcal{F}$ is the set of atomic formulas.
- $\mathcal{P}$ is the set of atomic formulas indexed by values from $I$.
- $I$ is the set of index values (a subset of $\mathbb{N}$).
- $S$ is a set of states.
- $R \subseteq S \times S$ is the transition relation.
- $L : S \rightarrow (\mathcal{F}(\mathcal{P}) \cup \mathcal{F}(I \times I))$ is the proposition labeling. We will write $A_c$ instead of $(A_c)$.
- $s_0$ is the initial state.

We extend the relation $\models$ to deal with indexed CTL* formulas as well:
1. $s \models A_c \iff A_c \in L(s)$.
2. $s \models \bigvee_i f(i) \iff \text{there exists a } c \in I \text{ such that } s \models f(c)$.

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

**Figure 4-1:** Example to Illustrate Restrictions on ICTL*

Even without the nexttime operator, this logic is too powerful; by nesting the operators $\wedge$ and $\vee$, it might still be possible to count the number of processes in a concurrent system. Suppose we take as our Knipke structure the global state graph for the concurrent program in Figure 4-1. The following formula sets a lower bound on the number of processes:

$\bigvee (A_i \wedge EF(B_i \wedge \bigvee (A_j \wedge EF(B_j \wedge \bigvee (A_k \ldots))))))$

Once $B_i$ becomes true, it remains true. Therefore, if $\bigvee A_k$ 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:
• \( \forall f \) is a permissible state formula only if \( f \) does not contain any \( \forall \) operators.
• \( \forall_i \bigwedge \exists \) is a permissible path formula only if neither \( \exists_i \) nor \( \forall_i \) contains any \( \forall \) 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 \( \forall \) 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 atomic formula, \( \bigoplus P \), to \( \bigwedge \) for every \( P \) in \( \bigwedge \). The proposition labeling is then extended as follows: \( \bigoplus P \in L(s) \) if and only if there is exactly one \( e \in I \) such that \( P_e \in L(s) \). 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. 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 from each structure at a time. Because of this, we will define a set of correspondence relations, \( F_{i'f} \), 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 follows:

\[
L_i(s) = \{ A | A \in \bigwedge (s') \} \cup \{ A | A \in \bigwedge (s) \}
\]

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

Now, we say that two structures, \( M_i \) and \( M_{i'} \), with the same set of indexed and nonindexed atomic formulas, \( (i,i') \) correspond if and only if \( M_i \) \( F_{i'f} \) \( M_{i'} \). We will write this as \( M_i \sim_{i'} M_{i'} \).

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 \( M_i \) and \( M_{i'} \) be two structures that \( (i,i') \) correspond. Let \( h(i) \) be an indexed ICTL formula without any \( \forall \) operators and with one free index variable. Let \( \pi \) be a path in \( M_i \) starting with \( s \) and \( \pi' \) be a path in \( M_{i'} \) starting with \( s' \). If there is a partition of \( \pi \) \( (B_1B_2\ldots) \) and a partition of \( \pi' \) \( (B'_1B'_2\ldots) \) such that all of the blocks are finite and \( B_f \sim_{i'f} B'_f \) for all \( f \) 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 $(v;v')$-correspondence, $s = A_i \Rightarrow s' = A_j$ is immediate.

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

**Theorem 5:** Let $M_i$ and $M_j$ be two structures and $IN$ be a relation over $I \times I$ that is total for both $I_i$ and $I_j$. If for every $(i,j) \in IN$, the two structures $(i;j')$-correspond, then $M_i. x_i \models h \Rightarrow M_j. x_j \models h$ 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 $h = \bigvee h_i(i)$. If $s_i \models \bigvee h_i(i)$, then there is some $i'$ such that $s_i \models h_i(i')$. Since $IN$ is total, there is an $i''$ such that $(i, i'') \in IN$. Therefore, since $M_i$ and $M_j (i, i'')$-correspond, Lemma 4 gives $s_j \models h_i(i'')$. Therefore, $s_j \models \bigvee h_i(i)$. The reverse argument is similar.

The proof of the remaining cases ($\neg h_i$ and $h_i \vee h_j$) 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 Figure 5-1. In the case of $r > 2$ processes, there may be more than one delayed process. Whenever this occurs, the process $P_j$ 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 $cln(i)$. We next define the state transition graph in the case of $r$ processes: $G = \langle AP, IP, P, S, R, L, S_0 \rangle$, where

- $AP = \emptyset$

- $IP = \{d,c,n,t\}$

---

1It is assumed that the token will be transferred through consecutive processes from $P_i$ to $P_{cln(i)}$ 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.
\[ I_r = \{1, \ldots, r\} \]

\[ S_r = \{s|s=\langle D.N.T.C.O \rangle\}, \text{ where} \]
- \[ D = \{i|s = d_i\} \]
- \[ N = \{i|s = n_i \wedge \neg t_i\} \]
- \[ T = \{i|s = n_i \wedge t_i\} \]
- \[ C = \{i|s = c_i \wedge t_i\} \]
- \[ O = I_r - (D \cup N \cup T \cup C) \]

We will refer to the sets \( D, N, T, C \) and \( O \) as the parts of state \( s \).

\[ R_r = \{(s,s')|s=\langle D.N.T.C.O \rangle \wedge s' = \langle D', N', T', C', O' \rangle \} \]

\[ \exists i \in N \wedge D_i = D \cup \{i\} \wedge N_i = N \setminus \{i\} \wedge T_i = T \wedge C_i = C \}\]

\[ \exists j \in D \wedge \exists t \in T \wedge C_i = \text{cin}(j) \wedge D_i = D \setminus \{i\} \wedge N_i = N \cup \{j\} \wedge T_i = T \setminus \{j\} \wedge C_i = (C \setminus \{j\}) \cup \{i\} \]

\[ \exists i \in T \wedge D_i = D \wedge N_i = N \wedge T_i = T \setminus \{i\} \wedge C_i = (C \cup \{i\}) \]

\[ \exists i \in C \wedge D = \emptyset \wedge D_i = D \wedge N_i = N \wedge T_i = T \cup \{i\} \wedge C_i = C \setminus \{i\} \]

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 \( P_i \) to a process \( P_j \), where \( i = \text{obt}(j) \). In the third transition a process with a the 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 the token, it remains with the same process.

\[
L_i(s) = \{ d_j | e D \} \cup \{ \eta_j | e \gamma \} \cup \{ d_j | e D \} \cup \{ \eta_j | e \gamma \}
\]

\[
\gamma' = \langle \emptyset, \{ 1 \} \rangle \emptyset \emptyset
\]

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.) It is easier to prove the correctness of the correspondence if we first show that certain simple invariants hold of our mutual exclusion program:

1. \( D, N, T, \) and \( C \) form a partition of \( I_r \); i.e. they are disjoint and \( O \) is always empty.
2. Once a process has requested the token, it will not stop requesting until the token is received.
   \[
   \land_i AG(a_i) \rightarrow \neg F[d_i U \neg d_i \land \neg t_i]
   \]
3. There is exactly one process with the token at any time. \( \land_i \Theta_i t_i \)

To establish these invariants, it is sufficient to show that they hold initially in \( \gamma' \) and every transition in \( R \), 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 \( \gamma' \), 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 transition to and from its critical section; therefore \( R_r \) is total.

Once we have established the correspondence using the invariants, we can apply the CTL model checking algorithm [4] to the two process mutual exclusion algorithm in order to establish the following properties:

1. A token is transferred only upon request.
   \[
   \neg \lor_i EF(\neg d_i \land \neg t_i \land \lor_i EF(\neg d_i \land \neg t_i U t_i))
   \]
2. Only the process with a token may get into its critical state.
   \[
   \land_i AG(t_i \rightarrow t_i)
   \]
3. Once a process has requested the token, it continues to request the token until the token is received.
\[ \bigwedge_i \forall G_i d_i = \forall i' | t_i' \]

4. Every process that wants to enter its critical state, eventually does.
\[ \bigwedge_j \forall G_j d_j = \forall c_j \]

In order to define the bisimulation between \( M_i \) and \( M_j \), we must first define the relation \( I \subseteq s_i \times s_j \), that determines the correspondence between index values in the two structures:
\[ I = \{ (i, j) \} \cup \{ (2, i) \mid i \in 1_n - \{ 1 \} \}. \]

Next, we must define the correspondence between states \( s_i \subseteq s_i \times s_j \times N \) for every \((i, j) \in I\):

1. Two states, \( s \) in \( M_i \) and \( s' \) in \( M_j \) (\( i, j \)) correspond if \( i \) is in the same part of \( s \) as \( i' \) is in \( s' \) and if \( i \in C \) then \( D = \emptyset \Rightarrow D' = \emptyset \).

2. Let an \( idle \) 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 \( D = \emptyset \) then \( D' = \emptyset \).

   We define the rank of \( s, r(s,i) \), to be the maximal number of consecutive \( idle \) 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 \( r(s,i) + r(s',i') \).

   Note that the only case in which the number of consecutive \( idle \) transitions from \( s \) is infinite is when \( s = n \). Also note that if \( s \) is reachable from \( s \) by pursuing \( idle \) transitions only and if \( r(s,i) \neq 0 \), then \( r(s,i) < r(s,i) \).

   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 \( idle \) 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 \( idle \) transitions in this case:

   a. Processes that are initially neutral may become delayed. \( (|N| - 1) \) transitions.

   b. The process with the token may enter its critical section. \( (|T| - 1) \) transitions.

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

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

   Therefore, \( r(s,i) = |N| + |T| + 2(j - i) \) mod \( n - 2 \).
3. \( i \in I \). The only idle transitions are neutral processes becoming delayed. So \( r(s,i) = \frac{1}{|N|} \).

4. \( i \in C \) and \( D = \emptyset \). 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 \( D \neq \emptyset \). The only idle transitions are neutral processes becoming delayed. Therefore, \( r(s,i) = \frac{1}{|N|} \).

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

Clause (1): Because all of the processes are neutral in the initial states of \( M_i \) and \( M_i' \) and process \( i \) has the token in each initial state, these states correspond for every \( (s,i) \in IN \), with a degree \( k = r(s,i) + r(s',i') \).

Clause (2a): Immediately from the definition of \( E_{P,i} \), for every two states \( s,s' \) that \( (s,i) \) correspond with any degree, \( s = A_i = s' = A_i' \) for every \( A \in IP \).

Clause (2b): Assume \( s \in E_{P,i} s' \) 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 \( i \) can become delayed in state \( s \). Since \( i' \in N \), process \( i' \) can also become delayed in some state \( s' \). These two next states are \( E_{P,i} \) related, since \( i \in D \) and \( i' \in D' \).

b. Some process can make an idle transition to state \( s \). In this case, some process in \( M_i \) can also make an \( i' \)-idle transition to \( s' \). Since \( i \) and \( i' \) are still in the same part, these two next states are \( E_{P,i} \) 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 idle transition to a state \( s \). Since \( i \in D \), \( s \in E_{P,i} s' \) for \( r(s,i) = r(s',i') \). \( r(s,i) \) measures the maximum possible number of idle transitions from \( s \). Because an idle transition from \( s \) has been made, \( r(s,i) < r(s,i) \) so \( r(s,i) < k \), 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 \( j' \) (\( i' \neq cln(j') \)). Thus, there must be a delayed process between \( j \) and \( j' \) 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, \( s' \), corresponds to \( s \) with degree 
\( r(s) = r(s) + r(s', s') \). Since an \( s' \) to \( s' \) transition from \( s' \) has been made, \( r(s', s') < r(s', s') \) so 
\( r(s') < k \), so clause (2b) holds.

**Clause (2c):** is proven similarly to clause (2b).

This completes the proof of the bisimulation of \( M_s \) and \( M_r \).

### 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 \( \wedge \)'s and \( \vee \)'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 \( \wedge \) and \( \vee \) 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.

We would like to acknowledge Prasad Sistla's insightful comments on an early version of this paper.

### References


END
FILMED
MARCH, 1988
DTIC