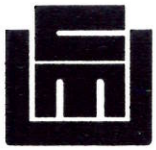
AUTOMATIC AND HIERARCHICAL VERIFICATION OF ASYNCHRONOUS CIRCUITS USING TEMPORAL LOGIC†

by
B. Mishra *and* E. M. Clarke

DEPARTMENT
of
COMPUTER SCIENCE



Carnegie-Mellon University

AUTOMATIC AND HIERARCHICAL VERIFICATION OF ASYNCHRONOUS CIRCUITS USING TEMPORAL LOGIC†

by
B. Mishra *and* E. M. Clarke
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

September, 1983

Abstract

Establishing the correctness of complicated asynchronous circuit is in general quite difficult because of the high degree of nondeterminism that is inherent in such devices. Nevertheless, it is also very important in view of the cost involved in design and testing of circuits. We show how to give specifications for circuits in a branching time temporal logic and how to mechanically verify them using a simple and efficient model checker. We also show how to tackle a large and complex circuit by verifying it hierarchically.

†.This research was partially supported by NSF Grant Number MCS-8216706.

Contents

0. Introduction.

Verification of the correctness of asynchronous circuits has been considered an important problem for a long time. But, a lack of any formal and efficient method of verification has prevented the creation of practical design aids for this purpose. Since all the known techniques of simulation and prototype testing are time-consuming and not very reliable, there is an acute need for such tools. Moreover, as we build larger and more complex circuits, the cost of a single design error is likely to become even higher. In this paper, we describe an automatic verification system for asynchronous circuits, in which the specifications are expressed in a propositional temporal logic. We illustrate the use of our system by verifying a version of the self-timed queue element given in [MC80].

Bochmann [BO82] was probably the first to recognize the usefulness of temporal logic to describe circuits; he verified an implementation of a self-timed arbiter using linear temporal logic and what he called "reachability analysis". The work of Malchi and Owicki [MO82] identified additional temporal operators required to express interesting properties of a circuit and also gave specifications of a large class of modules used in self-timed systems.

Although these researchers have contributed significantly toward developing an adequate notation for expressing the correctness of asynchronous circuits, the problem of mechanically verifying a circuit using efficient algorithms still remains unsolved. In this paper we show how a simple and efficient algorithm, called a *model checker*, can be used to verify various temporal properties of an asynchronous circuit. Roughly speaking, our method works by first building a labelled state-transition graph for an asynchronous circuit. This graph can be viewed as a finite *Kripke Structure*. Then by using the model checker we determine the truth of various temporal formulæ in this Kripke Structure. As a result, it is possible to avoid the complexity associated with proof construction.

Most complex circuits are built out of relatively less complex modules in a hierarchical manner. Hence it should be possible to verify these circuits in a hierarchical manner, *i.e.* to verify the correctness of a larger module, given the premises that the smaller modules are correct. A hierarchical approach to verification is important in practice, because it enables us to verify circuits incrementally, to localize faults to small submodules and most importantly, to handle large circuits without a large growth in complexity. We show how the hierarchical method can be incorporated in a mechanical approach to circuit verification.

The paper is organized as follows: Section 1 contains a brief description of the syntax and semantics of CTL, the temporal logic used in this paper, and also explains the algorithms used in the model checker. In Section 2, we give a simple step-by-step method used to verify circuits. In Section 3, we illustrate these methods by establishing some interesting properties of a Self-Timed Queue (FIFO) Element. In Section 4, we introduce a Hierarchical method to be used in verifying large and complex circuit and study some of the model-theoretic properties of the operation of "restriction" on a Kripke Structure. The paper concludes by pointing out the shortcomings of our approach and with a discussion of some remaining open problems.

1. CTL and Model Checker.

The logic that we use to give the specifications of a circuit is a propositional temporal logic of branching time, called CTL (Computation Tree Logic). This logic is essentially the same as that described in [CES83], [EC80] and [BMP81].

The syntax for CTL is given below:

Let $\mathcal{P}$ be the set of all the atomic propositions in the language, $\mathcal{L}$. Then

1. Every atomic proposition P in $\mathcal{P}$ is a formula in CTL.

2. If f_1 and f_2 are CTL formulæ, then so are $\neg f_1$, $f_1 \wedge f_2$, $\forall \mathbf{X} f_1$, $\exists \mathbf{X} f_1$, $\forall [f_1 \ \mathbf{U} \ f_2]$ and $\exists [f_1 \ \mathbf{U} \ f_2]$.

In this logic the propositional connectives $\neg$ and $\wedge$ have their usual meanings of negation and conjunction. The temporal operator $\mathbf{X}$ is the nexttime operator. Hence the intuitive meaning of $\forall \mathbf{X} f_1$ ($\exists \mathbf{X} f_1$) is that f_1 holds in every (in some) immediate successor state of the current state. The temporal operator $\mathbf{U}$ is the until operator. The intuitive meaning of $\forall [f_1 \ \mathbf{U} \ f_2]$ ($\exists [f_1 \ \mathbf{U} \ f_2]$) is that for every computation path (for some computation path), there exists an initial prefix of the path such that f_2 holds at the last state of the prefix and f_1 holds at all other states along the prefix.

We also use the following syntactic abbreviations:

$f_1 \vee f_2 \equiv \neg (\neg f_1 \wedge \neg f_2)$, $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$, and $f_1 \leftrightarrow f_2 \equiv (f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)$

$\forall \mathbf{F} f_1 \equiv \forall [\mathbf{true} \ \mathbf{U} \ f_1]$ which means for every path, there exists a state on the path at which f_1 holds.

$\exists \mathbf{F} f_1 \equiv \exists [\mathbf{true} \ \mathbf{U} \ f_1]$ which means for some path, there exists a state on the path at which f_1 holds.

$\forall \mathbf{G} f_1 \equiv\neg\ \exists \mathbf{F}\ \neg\ f_1$ which means for every path, at every node on the path f_1 holds.

$\exists \mathbf{G} f_1 \equiv\neg\ \forall \mathbf{F}\ \neg\ f_1$ which means for some path, at every node on the path f_1 holds.

$\forall[f_1\ \mathbf{W}\ f_2] \equiv\neg\ \exists[(f_1 \wedge f_2)\ \mathbf{U}\ (\neg\ f_1 \wedge f_2)]$ which means that for every computation path, and for every initial prefix of the path, if f_2 holds at all the states along the prefix then f_1 holds at all the states along the same prefix.

$\exists[f_1\ \mathbf{W}\ f_2] \equiv\neg\ \forall[(f_1 \wedge f_2)\ \mathbf{U}\ (\neg\ f_1 \wedge f_2)]$ which means that for some computation path, and for every initial prefix of the path, if f_2 holds at all the states along the prefix then f_1 holds at all the states along the same prefix.

In the last two formulæ $\mathbf{W}$ is the *while* operator. The formula $\forall[f_1\ \mathbf{W}\ f_2]$ ($\exists[f_1\ \mathbf{W}\ f_2]$) is read as "for every (some) path f_1 *while* f_2".

The semantics of a CTL formula is defined with respect to a labelled state-transition graph. A CTL structure is a triple $M = (S, R, \Pi)$ where

1. S is a finite set of states.

2. R is a total binary relation on S ($R \subseteq S \times S$) and denotes the possible transitions between states.

3. Π is an assignment of atomic proposition to states, i.e. $\Pi : S \mapsto 2^P$.

A *path* is an infinite sequence of states $(s_0, s_1, s_2, \ldots)$ such that $\forall_i[\langle s_i, s_{i+1}\rangle \in R]$. For any structure $M = (S, R, \Pi)$ and state $s_0 \in S$, there is an *infinite computation tree* with root labelled s_0 such that $s \rightarrow t$ is an arc in the tree *iff* $\langle s, t\rangle \in R$.

The truth in a structure is expressed by $M, s_0 \models f$, meaning that the temporal formula f is satisfied in the structure M at state s_0. The semantics of temporal formulæ is defined inductively as follows:

$s_0 \models P$ *iff* $P \in \Pi(s_0)$.

$s_0 \models\neg\ f$ *iff* $s_0 \not\models f$.

$s_0 \models f_1 \wedge f_2$ *iff* $s_0 \models f_1$ and $s_0 \models f_2$.

$s_0 \models \forall \mathbf{X} f_1$ *iff* for all states t such that $\langle s_0, t\rangle \in R, t \models f_1$.

$s_0 \models \exists \mathbf{X} f_1$ *iff* for some state t such that $\langle s_0, t\rangle \in R, t \models f_1$.

$s_0 \models \forall[f_1\ \mathbf{U}\ f_2]$ *iff* for all paths $(s_0, s_1, s_2, \ldots), \exists_{i \geq 0}[s_i \models f_2 \wedge \forall_{0 \leq j < i}[s_j \models f_1]]$.

$s_0 \models \exists[f_1\ \mathbf{U}\ f_2]$ *iff* for some path $(s_0, s_1, s_2, \ldots), \exists_{i \geq 0}[s_i \models f_2 \wedge \forall_{0 \leq j < i}[s_j \models f_1]]$.

From these it is quite easy to see that the semantics of **U**, the until operator can be easily given in terms of a *least fixed-point* characterization:

$\forall[f_1\ \mathbf{U}\ f_2] \equiv \mu\mathcal{F}.f_2 \vee (f_1 \wedge \forall\mathbf{X}\mathcal{F})$.

$\exists[f_1\ \mathbf{U}\ f_2] \equiv \mu\mathcal{F}.f_2 \vee (f_1 \wedge \exists\mathbf{X}\mathcal{F})$.

The Model Checker for CTL can now be thought of as an algorithm that determines the satisfiability of a given temporal formula f_1 in a model M, by computing these fixed points. A full description of the algorithm is given in [CES83].

In order to determine if a CTL formula f is true in a structure $M = (S, R, \Pi)$, the algorithm labels each state of S so that when the algorithm terminates, the label of each state $s \in S$, *label*(s), will be equal to $\{f' \in sub(f) \mid M, s \models f'\}$, where each element of *sub*(f) is either a subformula of f or the negation of the subformula. Hence $M, s \models f$ *iff* $f \in$ *label*(s) at the termination of the algorithm.

The labelling algorithm works in several stages. In the i^{th} stage the algorithm labels the states by the subformulæ of length i. The labels assigned in the earlier stages, corresponding to the subformulæ of length less than i are used to perform the labelling in this stage. It can be shown that the algorithm makes at most $n = |f|$ stages of computation and that the total amount of the work involved in each stage is $O(\|S\| + \|R\|)$. Hence the time complexity of the Model Checker is $O(|f| \cdot (\|S\| + \|R\|))$. The algorithm is also fairly simple, since it involves only a few straightforward graph theoretic algorithms.

2. Verification of Circuits.

Given a circuit to be verified, the steps involved in using the Model Checker to assert the correctness of the temporal specifications are as follows:

Step 1. Building the Model.

The structure associated with the circuit is essentially a finite state-transition graph, with its vertices corresponding to the distinct states and the edges corresponding to the (possibly nondeterministic) transition between the states. The initial label associated with each state is the set of propositions true in that state. This labelled state-transition graph can be built using the following simple algorithm:

```
begin
    L := {initial state};
    while L ≠ ∅ do
    choose a state, say s from L and delete it from L;
        for all sets of inputs, possible in s do
            simulate s with this set of inputs;
            let L' be the set of new states;
            for each s' ∈ L' do
                s' is a successor of s;
                if s' has not been visited then
                    add s' to L;
            end;
        end;
    end;
end.
```

Algorithm 2.1

The Algorithm to build the Kripke Structure for an Asynchronous Circuit.

Step 2. Giving the Specifications of the Circuit in CTL.

This corresponds to the specifications of the temporal behaviour of the circuit. It usually involves *structural properties* (*i.e.* the specifications for different components of the circuit, specifications of the signalling scheme used for communication with various other modules, etc.), *safeness properties* and *liveness properties.* It should probably be pointed out that one need not give the complete specification of the circuit in order to verify some selected properties of the circuit using the model checker.

Step 3. Verifying the Circuit using the Model Checker.

This step involves the model checker which checks the truth of the specification (a formula in CTL) in the structure constructed in the *step 1.* The working of the Model Checker is described in the previous section.

3. Extended Example.

We illustrate the ideas presented so far by verifying some interesting properties of an asynchronous circuit. The example chosen for this purpose is one element of a Self-timed (FIFO) Queue , which originally appeared in an article by C. Seitz on self-timed system [MC80].

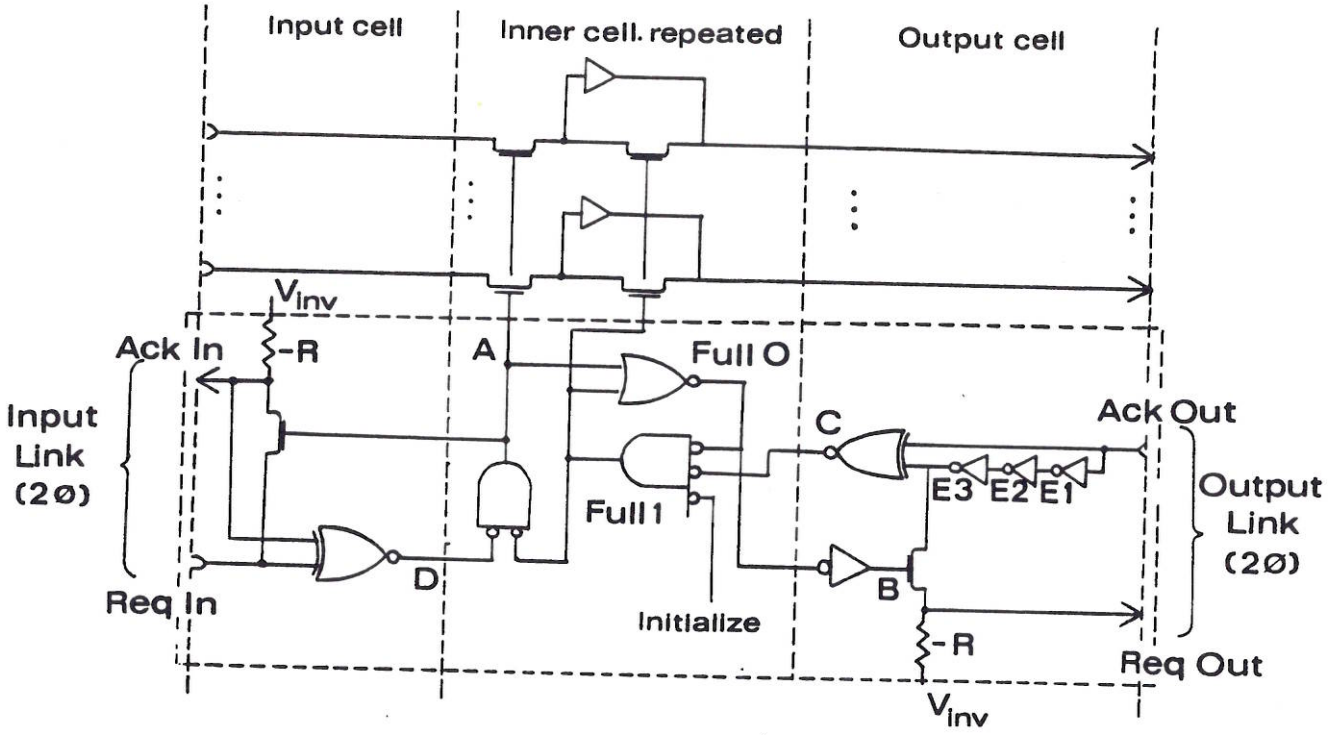


Figure. 3.1.
Queue (FIFO) element

a. Self-Timed FIFO Queue Element: The electrical circuit shown in *figure.* 3.1 is an implementation of a single FIFO queue element combined with some input and output logic. This circuit is of very practical importance; in pipeline processes in which operation times are variable, increased throughput can be achieved by interconnecting the processing elements through queues. The implementation uses simple asynchronous control and hence, can be used to build very fast and area-efficient queues.

The *inner cell* is intended to be replicated as many times as the number of words the queue is to be able to store, and the same control will operate a queue of any word length. The *input cell* and the *output cell* can be thought of as logic circuits converting the *two-cycle* signalling scheme at the input link to a *four-cycle* signalling scheme at the internal link and *vice versa*. The *inner cell* can be thought of as a latch that stores the state of the cell (*i.e.* whether the cell is *full* or *empty*), together with logic to generate a *load* signal and a set of *static registers* to store the *bits*. However, the design shown is *not* speed-independent, and uses the 3/2-rules. That is one may expect misoperation if particular sets of 3 gates have a smaller cumulative propagation delay time than other sets of 2 gates.

In the following subsections we specify and verify some interesting properties of the Queue element with a single inner cell.

b. Temporal Specifications for the Self-Timed Queue Element: We give examples of the ways in which various properties of a circuit can be given in CTL. In case of the Queue Element some of the *structural properties* that we might like to specify, are that the two-cycle signalling used at the *input links* and the *output links* is safe and live. Recall that the structural properties are specifications for various components and signalling schemes and thus, may be considered as premises that must be true in any CTL structure modelling the circuit. Hence the *request* signal must satisfy the following *safeness* and *liveness* conditions. (In the following CTL specifications we will use symbols Req and Ack for the *request* and the *acknowledgement* signals respectively.)

Safeness Conditions for the Request Signal.

1. $\forall \mathbf{G}((\neg \mathrm{Req} \wedge \mathrm{Ack}) \rightarrow \forall[\neg \mathrm{Req}\, \mathbf{W}\, \mathrm{Ack}])$

2. $\forall \mathbf{G}((\mathrm{Req} \wedge \neg \mathrm{Ack}) \rightarrow \forall[\mathrm{Req}\, \mathbf{W}\, \neg \mathrm{Ack}])$

These two CTL formulæ essentially express that if the Req and Ack signals are non-equipotential then the Req signal will remain in its stable logic value while Ack signal is in its stable value. In other words, Req will not be given unless acknowledgement to previous request signal has arrived.

Liveness Conditions for the Request Signal.

1. $\forall \mathbf{G}((\mathrm{Req} \wedge \mathrm{Ack}) \rightarrow \forall \mathbf{F}(\neg \mathrm{Req}))$

2. $\forall \mathbf{G}((\neg \mathrm{Req} \wedge \neg \mathrm{Ack}) \rightarrow \forall \mathbf{F}(\mathrm{Req}))$

These two CTL formulæ express the property that if the Req and Ack signals are equipotential then *eventually* the Req signal will change its logic value, thus indicating an arrival of a request.

In a similar manner, we can specify the properties of the *response* signal.

Safeness Conditions for the Response Signal.

1. $\forall G((Req \wedge Ack) \rightarrow \forall[Ack\ W\ Req])$

2. $\forall G((\neg Req \wedge \neg Ack) \rightarrow \forall[\neg Ack\ W\ \neg Req])$

Informally, they express the fact that Ack will not be given unless there has been a Req signal to cause it.

Liveness Conditions for the Response Signal.

1.$\forall G((Req \wedge \neg Ack) \rightarrow \forall F(Ack))$

2.$\forall G((\neg Req \wedge Ack) \rightarrow \forall F(\neg Ack))$

That is, if there had been a Req signal then *eventually* there will be an Ack signal in response to the request.

We can also give the *safeness* and the *liveness properties* of the FIFO Queue element in CTL. The following is a representative list of some of the properties, and by no means, exhaustive and complete. In the CTL formulæ given below, ReqIn stands for *request* at the *input links*, AckIn, for *acknowledgement* at the *input links*, ReqOut, for *request* at the *output links*, AckOut, for *acknowledgement* at the *output links* and Full1, for the state of the queue element when it holds some data.

Some Safeness Properties of the Queue Element.

1.$\forall G(\neg (ReqIn = AckIn) \wedge \neg (ReqOut = AckOut) \rightarrow \forall[\neg (ReqIn = AckIn)\ U\ (ReqOut = AckOut)])$

This formula states that if there have been a ReqIn and a ReqOut, then AckIn will not be given until AckOut has arrived.

Some Liveness Properties of the Queue Element.

1.$\forall G(\neg (ReqIn = AckIn) \wedge \neg Full1 \rightarrow \forall F(A))$

This formula states that if there has been a ReqIn, and the memory element was empty, then *eventually* it will be loaded with the input data.

2.$\forall$G(Full1$\rightarrow$ $\forall$F($\neg$ (ReqOut=AckOut)))

That is the Queue Element is full then *eventually* a request at the *output links* will be generated in order to move the data to the next element in the queue.

3.$\forall$G((ReqOut=AckOut) $\rightarrow$ $\forall$F($\neg$Full1))

That is if the acknowledgement arrives at the *output links* thus indicating that the data stored in the current Queue Element has been moved to the next element, then *eventually* the Queue Element will mark its state as empty.

In the next subsection we show how these specifications can be verified automatically by using a Model Checker.

c. Verification of the Circuit: As a first step for the verification of the circuit, we build a labelled finite state-transition graph corresponding to the circuit given in *figure.* 3.1, using the algorithm given in section 2. For this model, we assume that each gate of the circuit has *one unit delay.* This is done in order to take care of the speed-dependent properties of the circuit. This is equivalent to assuming that for any state in the graph, any of the successor states is arrived at after one unit gate-delay. The label associated with each state is the set of nodes in the circuit which assume the logical value 1 in that state. The nodes of the circuit are — AckIn, ReqIn, D, A, Full0, Full1, C, B, E1, E2, E3, ReqOut and AckOut. The initial state corresponds to the situation when ReqIn and AckIn as well as ReqOut and AckOut are equipotential.

Now,the model checker can take a description of the model and a temporal formula specifying some property of the circuit, and determine truth of the formula in that model. However the circuit shown does *not* obey the 3/2 rule as advertised, and the model checker determines that the safeness property of the queue element, given in the previous subsection is *not true.*

Informally, the problem can be described as follows: When an AckOut is received in response to the ReqOut signal, the AckOut signal travels via two different electrical paths — one involving three inverters and the other involving four gates. This creates a race condition and produces a glitch of about one gate delay on the ReqOut bus. Though this glitch may not always be able to drive the bus to create a spurious ReqOut, it has the potential to do so. However, this problem can be easily rectified by making the inverters slow or by putting five inverters on that path instead of three. The labelled state-transition graph for the corrected circuit is shown in *figure.* 3.2.

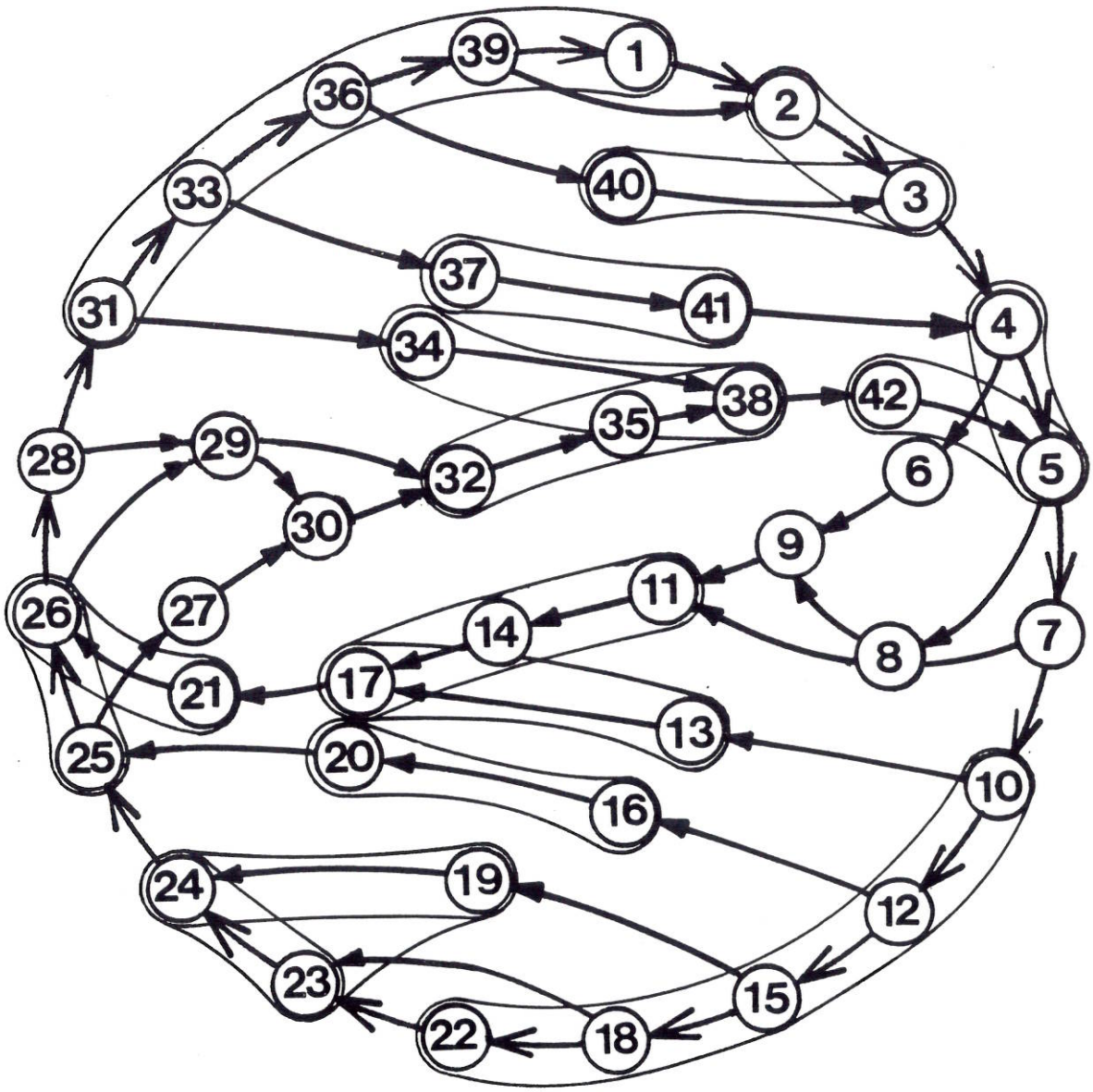


Figure. 3.2.
The State-Transition Graph for the Self-Timed Queue Element

The state-transition graph shown in *figure.* 3.2. is only one portion of the complete state-transition graph for the FIFO Queue Element and corresponds to the initial state where both ReqIn and AckIn are both at logical-zero value and both ReqOut and AckOut are at logical-zero value. But the state in which both ReqIn and Ackin are at logical-zero and both ReqOut and AckOut are at logical-one can not be reached from this state-transition graph. In fact the state-graph with this situation as the initial condition is symmetric to the one shown and the complete state-transition graph consists of both of these components.

```
time:  (1453 168)
|= AG(((~ ReqIn & AckIn) | (ReqIn & ~ AckIn)) &
((~ ReqOut & AckOUt) | (ReqOut & ~ AckOut)) ->          [ <  7 secs.]
A[((~ ReqIn & AckIn) | (ReqIn & ~ AckIn)) U
((ReqOut & AckOut) | (~ ReqOut & ~ AckOut))])

t

time:  (2263 300)
|= AG( ((~ ReqIn & AckIn) | (ReqIn & ~ AckIn)) & (~ Full1) -> AF(A))
                                                        [ < 8 secs.]
t

time:  (2694 300)
|= AG(Full1 -> AF( ((~ ReqOut & AckOut) | (ReqOut & ~ AckOut))))
                                                        [ < 8 secs.]
t

time:  (3150 300)
|= AG(((ReqOut & AckOut) | (~ ReqOut & ~ AckOut)) -> AF(~ Full1))
                                                        [ < 7 secs.]
t
```

Figure. 3.3
A sample run using the Model Checker.

A sample run using the model checker is shown in *figure.* 3.3. In the formula shown `A` stands for $\forall$, `E` for $\exists$, `|` for $\vee$, `&` for $\wedge$, `~` for $\neg$ and `->` for $\rightarrow$. Similarly, `G`, `F`, `U` and `W` will stand for **G**, **F**, **U** and **W**, respectively. The first component of "`time:`" is the cumulative time in 60^{th} of a second; the second component is the portion of the cumulative time allocated to 'garbage collection'. The number to the right of each formula gives the time taken to determine the truth of the formula.

4. Hierarchical Verification of Circuits.

The scheme given so far can be practical only for very small circuits. This is because it suffers from the problem that the state transition graph may have number of states, exponential in number of gates. However, this problem can be avoided, if circuits are verified in a hierarchical manner. That is, first small modules are verified and then bigger module is verified assuming that the smaller modules it is composed of are correct. Since at any hierarchical level, the number of small modules that a big module is composed of is relatively small, this method is amenable to proving correctness of large circuits without a large growth of the time complexity. Moreover, hierarchical verification permits the localization of faults to small submodules, thus allowing the designer to rectify the fault by redesigning the appropriate submodule.

In a hierarchical approach, the state transition graph for a circuit is built out of the descriptions of the constituent submodules. We obtain short a description of a module by using an operation called 'restriction'. If $\mathcal{L}$ is the language for the module with a set of atomic propositions P, corresponding to the input, output and internal nodes, then the operation restriction on $\mathcal{L}$, obtains a $\mathcal{L}'$ with atomic propositions P', corresponding to the input and the output nodes only.

Roughly speaking, the effect of restriction is to make the internal nodes invisible, since in building the state transition graph for the bigger module, we only require input-output behaviour of the constituent submodules. But when the internal nodes are made invisible, certain portions of the state graph will have same labelling of the atomic (input and output) propositions. The restriction operation defines exactly when such states can be collapsed into a single state.

Unfortunately, when we restrict a CTL structure to obtain a smaller structure, some formulæ that are true in the former structure may not be true in the restricted structure. However, by appropriately constraining CTL, we can show that the formulæ in the constrained logic have the desirable property that the truth properties of such formulæ are preserved with respect to the restriction operation. All of the formulæ used in *section 3.* have the desired syntax.

Let the CTL structure for $\mathcal{L}$ be $\mathcal{M} = (S, R, \Pi)$. Let $\mathcal{P}$ be the set of all atomic propositions in the language $\mathcal{L}$, consisting of $\mathcal{I}$, the set of atomic propositions corresponding to the *inputs*; $\mathcal{O}$, the set of atomic propositions corresponding to the *outputs* and *Int*, the set of atomic propositions corresponding to the *internal nodes* of the circuit. That is $\mathcal{P} = \mathcal{I} \cup \mathcal{O} \cup Int$. Let $\mathcal{L}'$ be the language with the atomic propositions, $\mathcal{P}' = \mathcal{I} \cup \mathcal{O}$. Define $\Pi_{\mathcal{P}'} : S \mapsto 2^{\mathcal{P}'}$ to be the restriction of Π to $\mathcal{P}'$, *i.e.* $\forall_{s \in S}[\Pi_{\mathcal{P}'}(s) = \Pi(s) \cap \mathcal{P}']$. Now we can define a relation $\mathcal{E}$ ($\mathcal{E} \subseteq S \times S$) over the set of states of $\mathcal{M}$ such that

$s \mathcal{E} s'$ *iff* for some path $(s_0, s_1, \ldots, s_n)$ of $\mathcal{M}$, $n \geq 0$, $s = s_0$ and $s_n = s'$ and for each predecessor of s_i, s_i' $(1 \leq i \leq n)$, $\Pi_{\mathcal{P}'}(s_i') = \Pi_{\mathcal{P}'}(s_i)$.

It is quite easy to see that the relation $\mathcal{E}$ over S, is *reflexive* and *transitive* but not *symmetric*. The transitive closure of $\mathcal{E}$ can be defined as

$$\mathcal{E}^* = \mathcal{E} \cup \mathcal{E}^2 \cup \mathcal{E}^3 \cup \ldots \cup \mathcal{E}^n \cup \ldots$$

The $\mathcal{E}$-*closure* of a state s is defined by $\mathcal{E}^*(s) = \{s' \mid s \mathcal{E}^* s'\} = \{s' \mid s \mathcal{E} s'\}$, since $\mathcal{E}$ is a transitive relation, *i.e.* $\mathcal{E}^* = \mathcal{E}$.

For a set of sets $\{u_j\}$, $\max(\{u_j\})$ will denote the set of all distinct sets in $\{u_j\}$ maximal under inclusion. We define a mapping $\varphi : S \mapsto 2^S$ such that for each $s \in S$,

$$\varphi(s) = \max(\{H_i \mid s \in H_i \wedge \exists_{s_i \in S} \mathcal{E}^*(s_i) = H_i\}),$$

i.e. $\varphi(s)$ is the set of maximal $\mathcal{E}$-closures containing s. We consider the following subsets of S,

$$\Delta = \varphi(S) = \bigcup_{s \in S} \varphi(s).$$

Since every element $s \in S$ belongs to at least one subset H_i of Δ, Δ is called a *decomposition* of S and the H_i's are called the *blocks* of the decomposition. We will say s *dominates* s', if $s \mathcal{E} s'$. We define the *dominant states* of H_i, $\mathrm{dom}(H_i)$ as the set of states that dominate every other states in H_i.

The decomposition Δ naturally leads to a *substructure* of a model $\mathcal{M}$ (notation $\mathcal{M}' = (S', R', \Pi') = \mathcal{M}/\Delta$). The states of $\mathcal{M}'$ will be the blocks of Δ. A block H_i of Δ, when considered as an element of S', will be denoted by $\overline{H}_i$. Let R' ($R' \subseteq S' \times S'$) be the total binary relation on S', corresponding to R and induced by the decomposition Δ *i.e.*

$\langle \overline{H}_i, \overline{H}_j \rangle \in R'$, for $i \neq j$ iff for some $s_i \in H_i, s_j \in H_j, \langle s_i, s_j \rangle \in R$ and $s_j \notin H_i$.

$\langle \overline{H}_i, \overline{H}_i \rangle \in R'$ iff for some $s_i, s_j \in H_i, s_j \, \mathcal{E} \, s_i$ and $\langle s_i, s_j \rangle \in R$.

Similarly, let $\Pi' : S' \mapsto 2^{\mathcal{P}'}$ be the mapping corresponding to Π and induced by the decomposition Δ, *i.e.*

$$\Pi'(\overline{H}_i) = \mathcal{P}' \cap \bigcap_{s \in H_i} \Pi(s).$$

The model $\mathcal{M}' = (S', R', \Pi')$ is called a *restriction* of $\mathcal{M} = (S, R, \Pi)$ with respect to $\mathcal{P}' \subseteq \mathcal{P}$.

In the following theorem, we show that there are CTL formulæ whose truth-properties are not preserved with respect to restriction.

THEOREM 4.1. *There exists a CTL structure $\mathcal{M} = (S, R, \Pi)$ and a formula $\mathcal{F}$ where $\mathcal{F}$ is a CTL formula such that*

$$\mathcal{M}, s_0 \models \mathcal{F} \qquad \textit{but} \qquad \mathcal{M}', \overline{H}_0 \not\models \mathcal{F}, \qquad \textit{and } s_0 \in \textit{dom}(H_0).$$

Proof. We give counter-examples involving formulæ of the form $\forall \mathbf{X} P$, $\exists \mathbf{X} P$ and $\forall[\exists \mathbf{F} P_1 \, \mathbf{U} \, P_2]$.

We first give a model $\mathcal{M} = (S, R, \Pi)$ over a language $\mathcal{L}$ such that $\mathcal{M}, s_0 \models \forall \mathbf{X} P$ and $\mathcal{M}, s_0 \models \exists \mathbf{X} P$, but $\mathcal{M}', \overline{H}_0 \not\models \forall \mathbf{X} P$ and $\mathcal{M}', \overline{H}_0 \not\models \exists \mathbf{X} P$, where $\mathcal{M}'$ is a restriction of $\mathcal{M}$ and $s_0 \in \mathrm{dom}(H_0)$.

Define $\mathcal{M} = (S, R, \Pi)$ over a language $\mathcal{L}$ with the set of propositions $\mathcal{P}$,

$$\begin{aligned} \mathcal{P} &= \{P_{in}, P'_{in}, P_{int}\} \text{ and} \\ S &= \{s_0, s_1, s_2\} \text{ and} \\ R &= \{\langle s_0, s_1 \rangle, \langle s_1, s_2 \rangle, \langle s_2, s_2 \rangle\} \end{aligned}$$

and Π to be $\Pi(s_0) = \{P_{in}, P_{int}\}$, $\Pi(s_1) = \{P_{in}\}$ and $\Pi(s_2) = \{P'_{in}, P_{int}\}$. Clearly, $\mathcal{M}, s_0 \models \forall \mathbf{X} P_{in}$ and $\mathcal{M}, s_0 \models \exists \mathbf{X} P_{in}$. Now if we take restriction of $\mathcal{M}$ for language $\mathcal{L}'$ with the set of propositions $\mathcal{P}'$,

$$\mathcal{P}' = \{P_{in}, P'_{in}\},$$

then we get $\mathcal{M}' = (S', R', \Pi')$ where

$$\begin{aligned} S' &= \{\overline{H}_0, \overline{H}_1\}, \\ R' &= \{\langle \overline{H}_0, \overline{H}_1 \rangle, \langle \overline{H}_1, \overline{H}_1 \rangle\} \end{aligned}$$

and Π' to be $\Pi'(\overline{H}_0) = \{P_{in}\}$ and $\Pi'(\overline{H}_1) = \{P'_{in}\}$. It can be easily seen that $\mathcal{M}', \overline{H}_0 \not\models \forall \mathbf{X} P_{in}$ and $\mathcal{M}', \overline{H}_0 \not\models \exists \mathbf{X} P_{in}$.

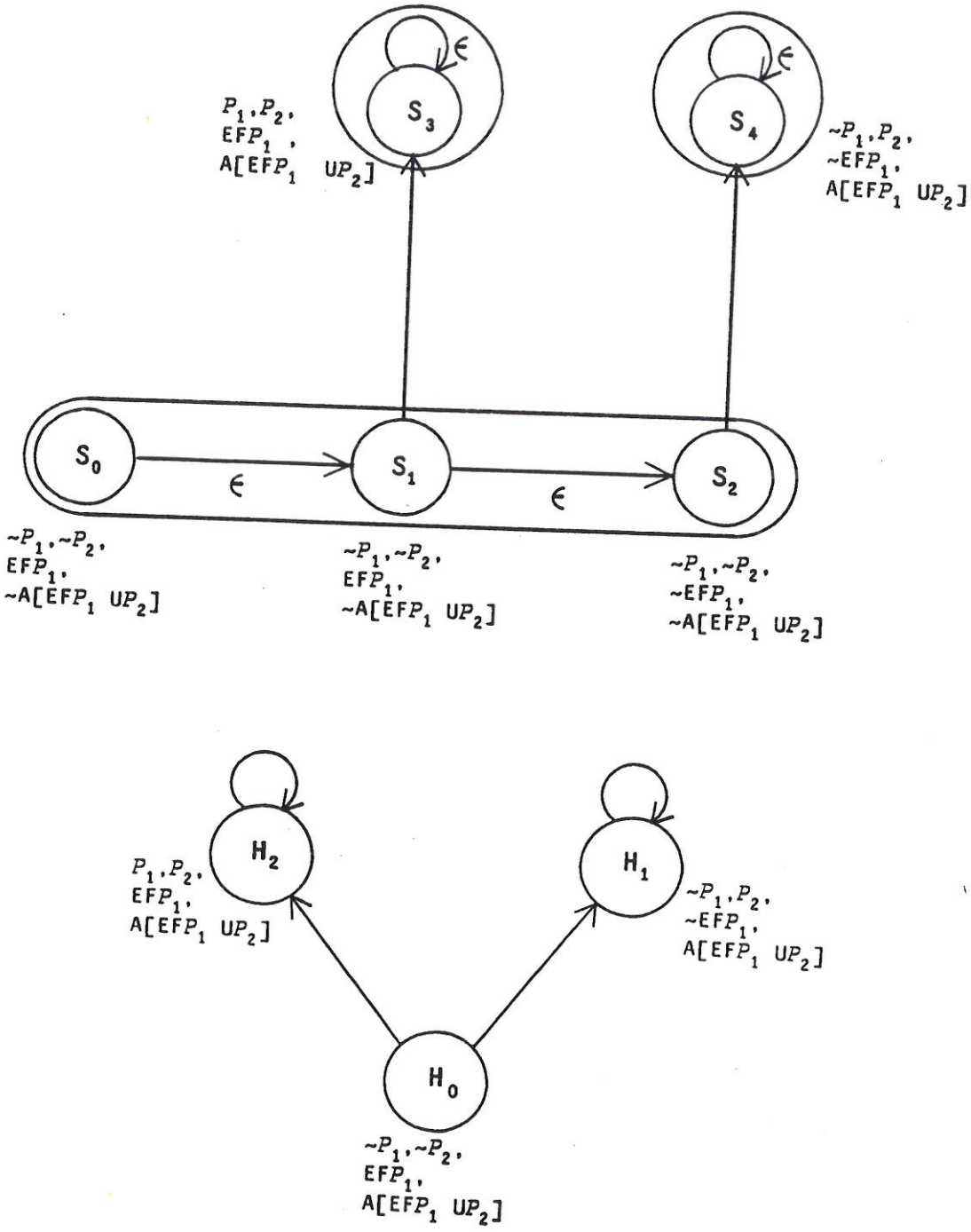


Figure. 3.2.
Counter-Example for Theorem 4.1.

Similarly, we present a model $\mathcal{M} = (S, R, \Pi)$ such that $\mathcal{M}, s_0 \models \neg\, \forall[(\exists \mathbf{F} P_1)\ \mathbf{U}\ P_2]$, but $\mathcal{M}', \overline{H}_0 \not\models \neg\, \forall[(\exists \mathbf{F} P_1)\ \mathbf{U}\ P_2]$, where $\mathcal{M}'$ is a restriction of $\mathcal{M}$ and $s_0 \in \mathrm{dom}(H_0)$.

Define $\mathcal{M} = (S, R, \Pi)$ over a language $\mathcal{L}$ with the set of propositions $\mathcal{P}$

$$\begin{aligned}
&\mathcal{P} = \{P_1, P_2, P_{int1}, P_{int2}\} \text{ and} \\
&S = \{s_0, s_1, s_2, s_3, s_4\} \text{ and} \\
&R = \{\langle s_0, s_1\rangle, \langle s_1, s_2\rangle, \langle s_1, s_3\rangle, \langle s_2, s_4\rangle, \langle s_3, s_3\rangle, \langle s_4, s_4\rangle\}
\end{aligned}$$

and Π to be $\Pi(s_0) = \{P_{int1}\}$, $\Pi(s_1) = \emptyset$, $\Pi(s_2) = \{P_{int2}\}$, $\Pi(s_3) = \{P_1, P_2\}$ and $\Pi(s_4) = \{P_2\}$. The labellings in figure 4.1 show that $\mathcal{M}, s_0 \models \neg\, \forall[(\exists \mathbf{F} P_1)\ \mathbf{U}\ P_2]$.

Now if we take restriction of $\mathcal{M}$ for language $\mathcal{L}'$ with the set of propositions $\mathcal{P}' = \{P_1, P_2\}$, then we get $\mathcal{M}' = (S', R', \Pi')$ where

$$\begin{aligned}
&S' = \{\overline{H}_0, \overline{H}_1, \overline{H}_2\}, \text{ and} \\
&R' = \{\langle H_0, H_1\rangle, \langle H_0, H_2\rangle, \langle H_1, H_1\rangle, \langle H_2, H_2\rangle\}
\end{aligned}$$

and Π' to be $\Pi'(\overline{H}_0) = \emptyset$, $\Pi'(\overline{H}_1) = \{P_2\}$ and $\Pi'(\overline{H}_2) = \{P_1, P_2\}$. Now the labellings in figure 4.1 show that $\mathcal{M}', \overline{H}_0 \models \forall[(\exists \mathbf{F} P_1)\ \mathbf{U}\ P_2]$. ∎

However, there exists a large subclass of CTL formulæ with the desirable property that if a formula in this subclass is satisfiable in the unrestricted CTL structure, $\mathcal{M}$, then it is satisfiable in the CTL structure, $\mathcal{M}'$ obtained by restriction. We call this subclass CTL$^-$.

Given a set of atomic propositions $\mathcal{P}$:

1. Every atomic proposition $P \in \mathcal{P}$ is a propositional formula in CTL$^-$.

2. If f_1 and f_2 are propositional formulæ in CTL$^-$, then so are $\neg\, f_1$, $f_1 \wedge f_2$.

3. If f_1 is a propositional formula and f_2 is a CTL$^-$ formula, then $\forall[f_1\ \mathbf{U}\ f_2]$ and $\exists[f_1\ \mathbf{U}\ f_2]$ are CTL$^-$ formulæ.

THEOREM 4.2. *Let $\mathcal{F}$ be a CTL$^-$ formula in $\mathcal{L}'$. Then*

$$\mathcal{M}, s_0 \models \mathcal{F} \qquad \textit{iff} \qquad \mathcal{M}', \overline{H}_0 \models \mathcal{F}, \qquad \textit{where } s_0 \in \textit{dom}(H_0).$$

Proof. From lemma 4.3. and 4.4. (see appendix for statement and proof of the lemmas.) ∎

With each model M, one can associate an automaton such that its states and transitions are same as that of M, but the transitions are additionally labelled with the set of input signals that cause the transition and the set of output signals associated with the transition. let A and A' be the automata associated with the models M and M', respectively. It can be easily shown that the relation φ is a *weak homomorphism* of A onto A' and hence A' is a *covering* of A [GI68].The above result can be strengthened, if we notice that[‡]

$$\varphi^{-1} M^{A}_{\epsilon^*\sigma\epsilon^*} = M^{A'}_{\epsilon^*\sigma\epsilon^*}\varphi^{-1}, \quad \text{and}$$
$$\varphi^{-1} N^{A}_{\epsilon^*\sigma\epsilon^*} = N^{A'}_{\epsilon^*\sigma\epsilon^*}, \quad \text{and}$$
$$\varphi\varphi^{-1} \supseteq I_{S^A},$$

where M^A and $M^{A'}$ are the transition functions and where N^A and $N^{A'}$ are the output functions of the automata A and A', respectively.

THEOREM 4.3. *Let A and A' be the automata associated with the models M and M', respectively. Then the models M and M' are input-output equivalent in the sense that for a sequence of input signals, x,*

$$N^A_x \subseteq \varphi N^{A'}_x, \text{ and } \varphi^{-1} N^A_x = N^{A'}_x,$$

where N^A and $N^{A'}$ are the output functions of the automata A and A', respectively.

Proof. See appendix for a proof of the theorem. ∎

Hence we see that even if the operation of restriction does not preserve all the CTL formulæ, the restricted model is equivalent to the original model in terms of its behaviour. We show how to build M' from M in the following three steps. M' is essentially a restriction of M with additional optimizations and labelling of the transitions of the state-transition graph.

step 1. Relabel the vertices and the edges of the CTL structure M. (*a*) Label each state by the subset of the propositions involving only the inputs and the outputs of the module. (*b*) Label the edges between two states with the same set of atomic propositions, by ϵ.

step 2. Construct the blocks of M, by first determining the dominant states using a depth first search over the underlying graph. Build M' by replacing each block by a single state. The graph can be optimized further by collapsing the "indistinguishable nodes" (*i.e.* nodes with the same label and successor states) into single node.

step 3. Label the edges of the graph by the set of input signals that causes the transition and the set of output signals associated with the transition.

---

‡.We represent the composition of functions $\varphi_1 : D_1 \mapsto D_2$ and $\varphi_2 : D_2 \mapsto D_3$ by $\varphi_1\varphi_2 : D_1 \mapsto D_3$. The transition function is $M : \Sigma \mapsto (S \mapsto S)$ and the output function is $N : \Sigma \mapsto (S \mapsto \Theta)$.

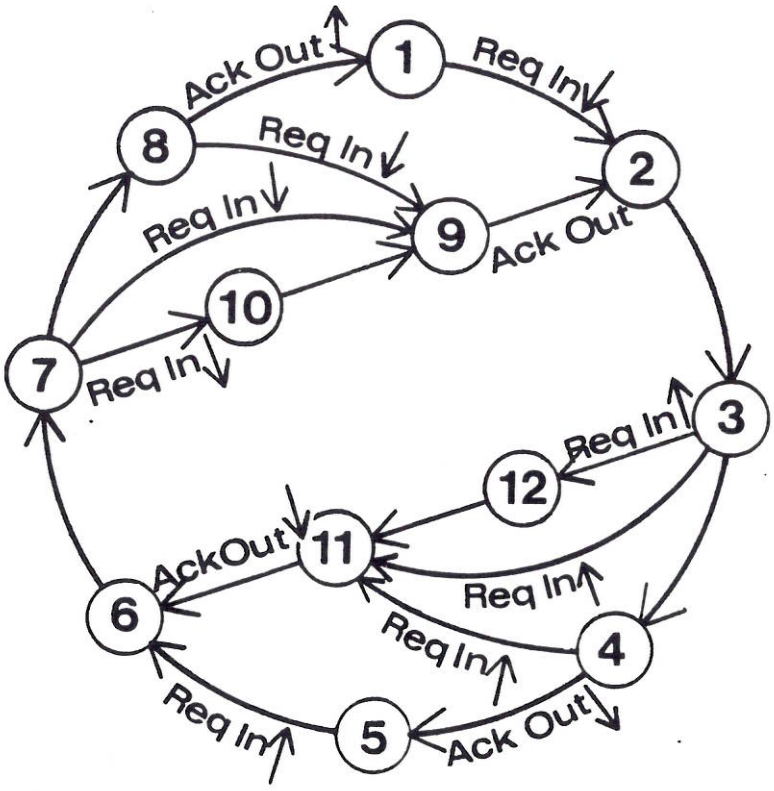


figure. 4.2.
The Restricted State Transition Graph.

This construction is illustrated by taking the restriction of the state-transition graph for the FIFO Queue Element shown in *figure.* 3.2. The states shown in groups are the blocks constructed in *step 2.* The resulting labelled state-transition graph is shown in *figure.* 4.2.

It should be mentioned that since we combine successive states in the operation of *step 2*, the restricted model may not be a unit-delay model even if the original unrestricted model was so. This notion is essentially captured in Theorems 4.1. and 4.2.

However, this does not pose a problem, since good design methodology forces the designer not to make the modules at higher level in the hierarchy speed-dependent. Moreover, since a speed-dependent circuits must be small enough to fit in an *equipotential region* and equipotential regions must be small enough that the potential on any wire in this area will equalize in a "short" time for any large circuit, the modules at higher level have to be speed-independent [MC80].

As the first step for verifying the correctness of a circuit using a hierarchical approach, we construct a CTL structure for a module at some hierarchical level, using the CTL structures for the submodules at the immediately lower level. In order to avoid building large-sized CTL structures, we use the restriction operation on the CTL structures of the submodules and obtain smaller descriptions of these. Moreover, the transitions of the state-transition graph are additionally labelled with the associated set of input signals and set of output signals, as explained earlier in this section.

Given two submodules A and B which are used to build a module C at a higher level by connecting the inputs and outputs of A and B, we show how to build a CTL structure for the module C using an operation called "composition". It can be shown that the composition operation is *commutative* and *associative* and hence can be generalized easily to the case where a module consists of more than two submodules. The reader may note a close analogy between the operations we define and the operations defined in [MI80].

Let the *restricted* models of the submodules A and B be $M_A = (S_A, R_A, \Pi_A)$ and $M_B = (S_B, R_B, \Pi_B)$, respectively. We assume that the propositions associated with A and B are renamed so that the input and output nodes of A and B that are connected have the same proposition associated with them. Furthermore, we make the important assumption that these connections are made using "*short*" *bilateral* wires.

The CTL structure of $C = A \circ B$ is given by $M_C = M_{A \circ B} = (S_{A \circ B}, R_{A \circ B}, \Pi_{A \circ B})$, where $S_{A \circ B} \subseteq S_A \times S_B$. The assignment function $\Pi_{A \circ B} : S_{A \circ B} \mapsto 2^{P_A \cup P_B}$ is defined by $\Pi(s_{A \circ B}) = \Pi(s_A) \cup \Pi(s_B)$ where the state $s_{A \circ B} = \langle s_A, s_B \rangle$. The initial state of M_C is $s_{0(A \circ B)} = \langle s_{0A}, s_{0B} \rangle$.

The transition relation $R_{A \circ B}$ ($R_{A \circ B} \subseteq S_{A \circ B} \times S_{A \circ B}$) is defined as follows. Assume that there is a transition $\langle s_{1A}, s_{2A} \rangle \in R_A$ such that $\langle s_{1A}, s_{2A} \rangle$ has associated with it, the input set α and the output set β. Similarly, assume that there is a transition $\langle s_{1B}, s_{2B} \rangle \in R_B$ such that $\langle s_{1B}, s_{2B} \rangle$ has associated with it the input set γ and the output set δ. Furthermore, assume that α is parttioned into disjoint subsets α' and α'' such that α' is associated with the inputs of C (*i.e.* the input transitions for α' are generated externally and the transitions for α'' are generated internally.) Similarly, assume that γ is parttioned into disjoint subsets γ' and γ''. Then in the CTL structure for C, there will be following transitions: (i) if $\alpha'' = \emptyset$, then there is a transition $\langle \langle s_{1A}, s_{1B} \rangle, \langle s_{2A}, s_{1B} \rangle \rangle \in$

$R_{A \circ B}$, with associated input α and output β; (ii) if $\alpha'' = \emptyset$, then there is a transition $\langle\langle s_{1A}, s_{1B}\rangle, \langle s_{1A}, s_{2B}\rangle\rangle \in R_{A \circ B}$, with associated input γ and output δ; and (iii) if (a) both $\alpha'' = \emptyset$ and $\gamma'' = \emptyset$, or (b) $\alpha'' \neq \emptyset$ and $\alpha'' \subseteq \delta$ or (c) $\gamma'' \neq \emptyset$ and $\gamma'' \subseteq \beta$, then there is a transition $\langle\langle s_{1A}, s_{1B}\rangle, \langle s_{2A}, s_{2B}\rangle\rangle \in R_{A \circ B}$, with associated input $\alpha \cup \gamma$ and output $\beta \cup \delta$.

The step of constructing the successor states for $\langle s_{1A}, s_{1B}\rangle$ can be thought of as simulating C at $\langle s_{1A}, s_{1B}\rangle$ for all possible sets of inputs and can be easily incorporated into algorithm 2.1. Now various properties of C with respect to the model M_C can be determined using the model checker algorithm, as explained in the earlier sections.

5. Conclusion.

We have shown that it is possible to do automatic verification of asynchronous circuit efficiently. We have also indicated how this method can be extended to do hierarchical verification of large and complex circuits. We believe that this approach may eventually turn out to be quite practical.

However, there are many problems that need to be addressed before this approach is made feasible in practice. In this paper we have used a unit-delay model for the circuit. Similarly, it is quite easy to use a steady-state model, in which each state in the state-transition graph corresponds to a stable state and only in response to an input change does a state change occur. While the steady-state model is useful for speed-independent self-timed circuits, the unit-delay model is needed to model properties of a speed-dependent circuit. Unfortunately, even for the speed-dependent circuits the assumption that each gate has one unit gate-delay is rather unrealistic, because two similar gates may have different delays depending on process variations, fan-outs of a gate etc. Moreover, because of various capacitive effects, the delay associated with a 0-to-1 transition is not equal to the one associated with a 1-to-0 transition. It is felt that it is necessary to find models that capture these properties better. Also, we do not know how to handle the effect of large fan-out, charge sharing etc. In addition, we felt that CTL is rather weak for succinctly expressing many properties of circuits. A notation based on temporal intervals [HMM83] may be more suitable for this purpose.

An interesting area for future research is the usefulness of restriction operation in the context of hierarchical verification. We have defined a "restriction" operation and shown that the truth-properties of the CTL^- formulæ are preserved with respect to the operation of restriction. It appears that any weaker version of "restriction" will not result in any substantial reduction of the size of the CTL structures and hence will make hierarchical verification rather expensive. On the other hand, it seems any stronger version of "restriction", will severely limit the class of CTL formulæ that will be preserved with respect to restriction.

6. Acknowledgement.

Thanks to Larry Rudolph of C-M. U. and Chuck Seitz of Caltech for helpful discussions.

7. References.

[BMP81] M. Ben-Ari, Z. Manna and A. Pnueli, "The Logic of Nexttime", *Eighth ACM Symposium on Principle of Programming Languages*, Williamsburg, VA, January 1981.

[BO82] G. V. Bochmann, "Hardware Specification with Temporal Logic: An Example", *IEEE Transactions on Computers*, Vol C-31,No. 3, March 1982.

[CES83] E. M. Clarke, E. A. Emerson and A. P. Sistla, "Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications: A Practical Approach", *Tenth ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1983.

[CM83] E. Clarke and B. Mishra, "Automatic Verification of Asynchronous Circuits", in *Proceedings of C-M.U. Workshop on Logics of Programs* (ed. E. Clarke and D.Kozen), Pittsburgh, PA, 1983 (to appear in Springer Lecture Notes in Computer Science).

[EC80] E. A. Emerson, E. M. Clarke, "Characterizing Properties of Parallel Programs as Fixpoints", *Proceedings of the Seventh International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science No. 85, Springer Verlag, 1981.

[GI68] A. Ginzburg, *Algebraic Theory of Automata*, Academic Press, New York . London, 1968.

[HMM83] J. Halpern, Z. Manna and B. Moszkowski, *A Hardware Semantics based on Temporal Intervals*, Report No. STAN-CS-83-963, Department of Computer Science, Stanford University, Stanford University, Stanford, CA 94305, March 1983.

[MC80] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Reading, MA, Addison-Wesley, 1980, Ch. 7.

[MI80] R. Milner, *A Calculus of Communicating Systems*, University of Edinburgh, June 1980.

[MO81] Y. Malchi and S. S. Owicki, "Temporal Specifications of Self-Timed Systems", in *VLSI Systems and Computations* (Ed. H. T. Kung, Bob Sproull, and G. Steele), Computer Science Press, 1981.

Let $\ell = (s_i, s_{i+1}, \ldots)$ be *any* path in $\mathcal{M}$ with $s_i \in \mathrm{dom}(H_i)$ and $\mathcal{R}_{p'}(\ell) = \ell' = (\overline{H}_i, \overline{H}_{i+1}, \ldots)$ be the corresponding path in $\mathcal{M}'$. By above, $\exists_{k \geq i} \mathcal{M}', \overline{H}_k \models f_2$. Let $p \geq i$ be the smallest index such that $s_p \in H_k$. Hence $s_p \in \mathrm{dom}(H_k)$. By *induction hypothesis* $\mathcal{M}, s_p \models f_2$. Since $\forall_{i \leq q < p} \exists_{i \leq l < k} s_q \in H_l$, and $\forall_{i \leq l < k} \mathcal{M}', \overline{H}_l \models f_1$ and f_1 is a propositional formula, we have $\forall_{i \leq q < p} \mathcal{M}, s_q \models f_1$. Hence using the semantics of the **U** operator, we get

$$\begin{aligned}
\mathcal{M}', \overline{H}_i \models \mathcal{F} &\Rightarrow \mathcal{M}', \overline{H}_i \models \forall[f_1 \ \mathbf{U} \ f_2] \\
&\Rightarrow \text{for all paths } (\overline{H}_i, \overline{H}_{i+1}, \ldots) \text{ of } \mathcal{M}', \\
&\qquad \exists_{k \geq i}[\mathcal{M}', \overline{H}_k \models f_2 \wedge \forall_{i \leq l < k}[\mathcal{M}', \overline{H}_l \models f_1]] \\
&\Rightarrow \text{for all paths } (s_i, s_{i+1}, \ldots) \text{ of } \mathcal{M}, \\
&\qquad \exists_{p \geq i}[\mathcal{M}, s_p \models f_2 \wedge \forall_{i \leq q < p}[\mathcal{M}, s_q \models f_1]] \\
&\Rightarrow \mathcal{M}, s_i \models \forall[f_1 \ \mathbf{U} \ f_2] \\
&\Rightarrow \mathcal{M}, s_i \models \mathcal{F}. \qquad \blacksquare
\end{aligned}$$

In the next lemma we will make use of following simple facts about a CTL$^-$ formulæ and blocks H_k, which we state without proof.

FACT 4.1. *If a state of H_k satisfies a propositional formula g, then all the states of H_k must satisfy g.* ∎

FACT 4.2. *Any quantified CTL$^-$ formula f_2 can be written in an expanded form*

$$\mathbf{Q}_1[g_1 \ \mathbf{U} \ \mathbf{Q}_2[g_2 \ \mathbf{U} \cdots \mathbf{Q}_n[g_n \ \mathbf{U} \ g_{n+1}] \cdots]]$$

where $\mathbf{Q}_1, \mathbf{Q}_2, \ldots \mathbf{Q}_n$ are path quantifiers $\forall$ or $\exists$, and $g_1, g_2, \ldots g_{n+1}$ are propositional formulæ. ∎

FACT 4.3. *If g_{n+1} holds in any state, so do the formulæ $\mathbf{Q}_j[g_j \ \mathbf{U} \ \mathbf{Q}_{j+1}[g_{j+1} \ \mathbf{U} \cdots \mathbf{Q}_n[g_n \ \mathbf{U} \ g_{n+1}] \cdots]]$ for all $1 \leq j \leq n$. Similarly, if $\mathbf{Q}_i[g_i \ \mathbf{U} \ \mathbf{Q}_{i+1}[g_{i+1} \ \mathbf{U} \cdots \mathbf{Q}_n[g_n \ \mathbf{U} \ g_{n+1}] \cdots]]$ holds in any state so do the formulæ $\mathbf{Q}_j[g_j \ \mathbf{U} \ \mathbf{Q}_{j+1}[g_{j+1} \ \mathbf{U} \cdots \mathbf{Q}_n[g_n \ \mathbf{U} \ g_{n+1}] \cdots]]$ for all $1 \leq j \leq i$. Conversely, if $\mathbf{Q}_1[g_1 \ \mathbf{U} \ \mathbf{Q}_2[g_2 \ \mathbf{U} \cdots \mathbf{Q}_n[g_n \ \mathbf{U} \ g_{n+1}] \cdots]]$ holds in some state then for some $1 \leq j \leq n$, g_j and $\mathbf{Q}_j[g_j \ \mathbf{U} \ \mathbf{Q}_{j+1}[g_{j+1} \ \mathbf{U} \cdots \mathbf{Q}_n[g_n \ \mathbf{U} \ g_{n+1}] \cdots]]$ hold in that state or g_{n+1} holds in that state.* ∎

LEMMA 4.4. *Let $\mathcal{F}$ be a CTL$^-$ formula in $\mathcal{L}'$. Then*

$$\mathcal{M}, s_i \models \mathcal{F} \quad \Rightarrow \quad \mathcal{M}', \overline{H}_i \models \mathcal{F}, \qquad \text{where } s_i \in \mathit{dom}(H_i).$$

Proof. We prove this by induction over a labelled computation tree, rooted at s_i and with branches corresponding to transitions in M. For the purpose of this proof we use an initial portion of the tree with the root at s_i, with branches corresponding to the transitions in block H_i and leaves corresponding to the dominating states of the blocks. Since $\mathcal{F}$ is in CTL$^-$, it is either of the form g or $\mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$, where g's are propositional formulæ. $M, s_i \models \mathcal{F}$, by assumption. We now label the tree as follows: if $\mathcal{F} = g$, then we label s_i with g. On the other hand, if $\mathcal{F} = \mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$, then depending on whether $\mathbf{Q}_1$ is $\forall$ ($\exists$), for all (some) computation paths starting from s_i, there exists an initial prefix of the path, such that $\mathbf{Q}_2[g_2\ \mathbf{U}\ \mathbf{Q}_3[g_3\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$ holds at the last state of the prefix and g_1 at all other states along the prefix. We label the states corresponding to the prefix with g_1 and continue the similar labelling procedure for all the last states of the prefix. Without loss generality we assume that $M, s_i \models g_1$.

This process stops when either some non-leaf is found to satisfy g_{n+1} or some leaf is reached and the leaf satisfies $\mathbf{Q}_j[g_j\ \mathbf{U}\ \mathbf{Q}_{j+1}[g_{j+1}\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$ for some $1 \leq j \leq n$. Let j be called the characteristic index of that state with respect to the formula $\mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$.

Basis Step: Either the formula $\mathcal{F}$ is of the form g or $\mathcal{F}$ is of the form $\mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$ and some non-leaf state of the initial portion of the computation tree satisfies g_{n+1}.

In the first case, since $M, s_i \models g$ and g is a propositional formula, it is easy to show that $M', \overline{H}_i \models g$. In the second case, by Fact 4.1. $M, s_i \models g_{n+1}$, and as in the first case, $M', \overline{H}_i \models g_{n+1}$. By Fact 4.3. $M', \overline{H}_i \models \mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$. Hence $M', \overline{H}_i \models \mathcal{F}$.

Induction Step: Formula $\mathcal{F}$ is of the form $\mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$ and g_{n+1} does not hold in any non-leaf state. Let k be the maximum over the characteristic indices of the leaves. Then there are two cases to consider:

Case A: $\mathbf{Q}_1, \mathbf{Q}_2, \ldots \mathbf{Q}_{k-1}$ are all $\forall$ quantifiers.

In this case all the leaves must satisfy $\mathbf{Q}_j[g_j\ \mathbf{U}\ \mathbf{Q}_{j+1}[g_{j+1}\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$ for some $1 \leq j \leq k$. By induction on computation tree, we have for the corresponding blocks H, $M', \overline{H} \models \mathbf{Q}_j[g_j\ \mathbf{U}\ \mathbf{Q}_{j+1}[g_{j+1}\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$. By Fact 4.3. $M', \overline{H} \models \mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$. But in the restricted structure M' , each of these $\overline{H}$ is a successor of $\overline{H}_i$. (By Lemma 4.1.(i)). Hence $M', \overline{H}_i \models \mathbf{Q}_1[g_1\ \mathbf{U}\ \mathbf{Q}_2[g_2\ \mathbf{U} \cdots \mathbf{Q}_n[g_n\ \mathbf{U}\ g_{n+1}]\cdots]]$. Hence $M', \overline{H}_i \models \mathcal{F}$.

Case B: $\mathbf{Q}_1, \mathbf{Q}_2, \ldots \mathbf{Q}_{k-1}$ are *not* all $\forall$ quantifiers. Assume $\mathbf{Q}_{i_1}, \mathbf{Q}_{i_2}, \ldots \mathbf{Q}_{i_p}$, $(1 \leq i_1 \leq i_2 \leq \cdots \leq i_p \leq k-1)$ are $\exists$ quantifiers.