Combining CTL, Trace Theory and Timing Models

Jerry R. Burch*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

A system that combines CTL model checking and trace theory for verifying speed-independent asynchronous circuits is described. This system is able to verify a large and useful class of liveness and fairness properties, and is able to find safety violations after examining only a small fraction of the circuit's state space in many cases. An extension has been implemented that allows the verification of circuits that are not speed-independent, but instead rely on assumptions about the relative delays of their components for correct operation. This greatly expands the class of circuits that can be automatically verified, making the verifier a more useful tool in the design of asynchronous circuits. The system is demonstrated on several fair mutual exclusion circuits, including a speed-independent version that is verified correct. It is also shown that given quite weak assumptions about the relative delays of components, the problem of designing a fair mutual exclusion circuit using a potentially unfair mutual exclusion element becomes almost trivial.

1 Introduction

1.1 Background

Much has been written about automatically verifying speed-independent asynchronous circuits using specifications expressed by formulas in the propositional temporal logic CTL [2,4,7]. A *model checker* has been implemented that verifies that a circuit satisfies a CTL formula, in time linear in both the length of the formula and the number of states of the circuit. CTL can express a wide range of specifications, including a large and useful class of liveness and fairness properties. However, some often used specifications, such as requiring that a circuit have no hazards, are tedious to express. Also, the entire state graph of a circuit must be constructed before any formulas can be checked. This can be a serious disadvantage when verifying a circuit with an error, since the state graph of

*This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, Amendment 20, under Contract Number F33615-87-C-1499, monitored by the Air Force Wright Aeronautical Laboratories.
The National Science Foundation also sponsored this research effort under Contract Number CCR-8722633.
The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

such a circuit is often several orders of magnitude larger than that of a similar correct circuit.

Another technique for verifying speed-independent asynchronous circuits is based on trace theory [5,6] as described by Dill. In this theory, a trace is a string of symbols, where each symbol is the name of a wire in the circuit being modeled. A trace represents a possible behavior of a circuit; the presence of a symbol in a trace represents a transition on the corresponding wire. All the possible behaviors of a circuit can be represented by a set of traces. Dill's technique differs from other theories based on traces by having *trace structures* that contain two sets of traces, a *success set* and a *failure set,* rather than just one set. Trace structures are used to represent both circuits and specifications. Dill described the implementation of an automatic verifier that checks a circuit against a specification in time linear in the number of states of both the specification and the circuit. Checking that a circuit has no hazards is a special case of checking that it has no *failures,* which is done automatically whenever a circuit is verified against a specification. Also, the trace theory verifier often finds errors in circuits after constructing only a small fraction of the states of the circuit. However, trace theory specifications cannot express any liveness or fairness properties.

CTL and trace theory based verification have been applied to only a very limited class of circuit timing models. Trace theory is only applicable to speed-independent circuits. Using such a conservative timing model forces sacrifices in circuit performance and complexity in order to assure correctness. In addition to the speed-independent model, CTL has been used with the unit delay model [2]. The unit delay model is quite liberal; it makes strong assumptions about the behavior of circuit components. It is possible, therefore, that circuits that are verified correct under the unit delay model may not actually work as desired in practice. Neither the speed-independent model or the unit delay model accurately represent the designers knowledge of the performance of circuit components. This limits the usefulness of verifiers based on these timing models.

1.2 New Results of this Paper

A verifier that combines CTL model checking and trace theory based verification has been implemented. It combines CTL's ability to express liveness and fairness properties with trace theory's ability to easily check for hazards and find safety violations without constructing a complete state graph.

We have extended trace theory in order to verify asynchronous circuits that are not speed-independent. This extension has been implemented in an automatic verifier. Various parameters of a gate's performance, including minimum and maximum delay, can be modeled. The potentially unbounded delays of devices in a metastable state can also be accurately modeled. Thus, circuits that depend on the relative delays of their components for correct operation can be accurately modeled and verified, making the verifier a useful tool in the design of these kinds of circuits.

One of the example circuits analyzed is a new speed-independent fair mutual exclusion circuit [3]. This circuit was designed using a variant of Martin's method for "compiling" circuits from CSP programs [10]. Using the verifier's ability to model timing assumptions, we show that such a complicated circuit is not necessary for practical fair mutual exclusion. Given reasonable timing assumptions, fair mutual exclusion can be implemented with a rather trivial circuit.

1.3 Outline of Paper

Sections 2 and 3 give a short summary of CTL and trace theory based verification. The combining of these two verification methods is described in Section 4. Two speed-independent fair mutual exclusion circuits are verified as examples. Section 5 describes extending the system to handle a wide range of timing models. The verification of two fair mutual exclusion circuits that use a variant of the 3/2 rule is described. Concluding remarks are in Section 6.

2 CTL and the Model Checker

CTL formulas are built up from a set of *atomic propositions.* We use AP to denote the set of atomic propositions, which in our context is equal to the set of wire names of the circuit being modeled. If an atomic proposition is true in a given state, that is interpreted as meaning that the corresponding wire is high in that state. The formal syntax of CTL is:

1. Every atomic proposition $p \in$ AP is a CTL formula.
2. If f_1 and f_2 are CTL formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, AXf_1, EXf_1, $A[f_1 \, U \, f_2]$, and $E[f_1 \, U \, f_2]$.

The symbols $\neg$ and $\wedge$ have the standard interpretations of logical negation and conjunction, respectively. X is the *nexttime* operator, the formula AXf_1 (EXf_1) intuitively means that f_1 holds in every (in some) immediate successor of the the current state. U is the *until* operator. A description of the meaning of U uses the notion of a *path,* which is an infinite sequence of states such that each state is followed by one of its immediate successors. The formula $A[f_1 \, U \, f_2]$ ($E[f_1 \, U \, f_2]$) intuitively means that for every path (for some 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 abbreviations in writing CTL formulas:

1. $AF(f) \equiv A[true \, U \, f]$ intuitively means that f holds in the future along every path; that is, f is *inevitable.*
2. $EF(f) \equiv E[true \, U \, f]$ intuitively means that there is some path that leads to a state in which f holds; that is, f *potentially* holds.
3. $EG(f) \equiv \neg AF(\neg f)$ means that there is some path on which f holds at every state.
4. $AG(f) \equiv \neg EF(\neg f)$ means that f holds at every state on every path; that is, f holds *globally.*

The semantics of CTL formulas are defined with respect to a labeled state-transition graph called a *CTL structure.* The nodes of this graph correspond to states of the circuit being represented. The truth of a CTL formula is relative to a CTL structure and a state in that structure. A formula may also be true relative to just a structure, in which case it is true relative to the structure and the start state designated in that structure. Each state is labeled with the set of atomic propositions that are true in that state. Each state also has associated with it the set of states that are its immediate successors.

In verifying circuits, we are often interested only in correctness along *fair* execution paths. For example, we may not wish to consider a path in which one gate fires infinitely often while another gate, which is firable, does not fire at all. This is handled by modifying the semantics of CTL so that all path quantifiers range over only fair paths. The fair paths of a particular circuit are specified by giving a list of CTL formulas, called *fairness constraints.* A path is fair if and only if for every fairness constraint, there are infinitely many states on the path that satisfy that constraint.

The Extended Model Checker (EMC) does automatic verification using CTL. A session with EMC starts with the loading of a description of the CTL structure of the circuit, along with any associated fairness constraints. Then CTL formulas are entered, and EMC checks whether they are satisfied by the circuit. For formulas that are not satisfied EMC outputs a counterexample path, if one exists, that illustrates why the formula does not hold.

3 Trace Theory

We can only give a short overview of trace theory in the space of this paper; the interested reader may refer to [5] and [6]. In trace theory based verification digital circuits and their specifications are modeled by *trace structures,* which are ordered 4-tuples of the form $T = (I, O, S, F)$. The set I is the set of names of the input wires of the circuit; O is the set of output wire names. The set $A = I \cup O$ is called the *alphabet* of T. The sets S (the *success set*) and F (the *failure set*) are regular sets of finite strings, called *traces,* over the alphabet A. Each trace models a possible behavior of the circuit by having each symbol in the trace represent a transition on the corresponding wire. This means that only the order of transitions is represented, not the actual times at which the transitions occur.

The S set and the F set are used to give partial specifications. A partial specification of a device describes requirements for the proper use of the device, and specifies the behavior of the device given that those requirements are satisfied. In a trace structure, the set S describes the behaviors of a device when it is used properly. The set F describes behaviors resulting from improper use. For example, in asynchronous circuits a gate is typically modeled so that the F set contains all behaviors that cause a hazard and the S set contains all behaviors that do not cause a hazard.

The set $P = S \cup F$ is the set of all *possible traces.* The S and P sets must be prefix-closed, and P must be non-empty. For some non-deterministic devices (such as the vending machine example in [11], which is also discussed in [5]) a given trace can be both a success and a failure, so the S and F sets need not be disjoint. Also, no circuit can control its inputs, which is modeled by requiring that $PI \subseteq P$ (if A and B are sets of strings or symbols, then $AB = \{ab : a \in A \text{ and } b \in B\}$). This is called the *receptiveness* requirement.

Trace theory cannot explicitly model simultaneous transitions, transitions are always modeled as occurring in some total order. This does not mean that simultaneous transitions are not allowed to occur in the circuits being modeled, however. An atomicity assumption is applied requiring that the simultaneous transition of two wires must have exactly the same effect on the state of the circuit as non-simultaneous transitions of the signals in either one order or the other, similarly for simultaneous transitions of more

than two wires. This assumption is quite reasonable for digital circuits, and it greatly simplifies the theory and the implementation of the verifier.

As an example, consider how a buffer might be modeled by a trace structure $T = (I, O, S, F)$. Let b be the name of the input wire of the buffer, and let x be the name of the output wire. Then $I = \{b\}$ and $O = \{x\}$. Assume that both the input and the output of the buffer are initially low. We define the set of successful behaviors of the buffer to be those behaviors in which the environment does not cause a hazard. Thus, the set S of successful behaviors is equal to $(bx)^*(b + \epsilon)$. If the environment does cause a hazard, then we do not restrict the ensuing behavior of the circuit, so $F = (bx)^* bb(b + x)^*$.

Consider two variations of the above buffer. The first buffer always eventually fires when it is put in a firable state. The second buffer sometimes might never fire. Both of these buffers would be modeled by exactly the same trace structure. This is an example of trace theory's inability to model liveness properties.

A composition operation can be defined on trace structures. Let $T_0 = (I_0, O_0, S_0, F_0)$ and $T_1 = (I_1, O_1, S_1, F_1)$ be trace structures. The composition of T_0 and T_1 is defined when O_0 and O_1 are disjoint. The set of outputs of the composition is $O_0 \cup O_1$, the set of inputs is $(I_0 \cup I_1) - (O_0 \cup O_1)$. Let $A_0 = I_0 \cup O_0$, $P_0 = S_0 \cup F_0$, $A_1 = I_1 \cup O_1$, and $P_1 = S_1 \cup F_1$. Also, let $A = A_0 \cup A_1$, i.e., the alphabet of the composition. If x is a trace in A^*, define the projection $x|_{A_0}$ to be the trace formed from x by removing all symbols not in A_0. The S and F sets of the composition are given by

$$\begin{aligned} S &= \{x \in A^* : x|_{A_0} \in S_0 \wedge x|_{A_1} \in S_1\}, \\ F &= \{x \in A^* : (x|_{A_0} \in F_0 \wedge x|_{A_1} \in F_1) \vee \\ &\quad (x|_{A_0} \in F_0 \wedge x|_{A_1} \in S_1) \vee \\ &\quad (x|_{A_0} \in S_0 \wedge x|_{A_1} \in F_1)\}. \end{aligned}$$

A trace structure (I, O, S, F) is said to be *failure free* when F is empty. If the composition of several trace structures is failure free, then the components have been composed in such a way that each of their environmental requirements has been satisfied. The trace theory verifier can efficiently check whether the composition of a set of trace structures is failure free. If a composition is not failure free, then usually only a small fraction of the state space needs to be explored before a short error trace can be given.

Let T_0 and T_1 be trace structures. Then, T_0 *conforms to* T_1 (written $T_0 \preceq T_1$) if $I_0 = I_1$ and $O_0 = O_1$ and for any trace structure T, the composition of T and T_1 being failure free implies that the composition of T and T_0 is failure free. The idea captured here is that if circuit works correctly with T_1 as a component, then it works correctly with T_1 replaced by T_0 (up to safety properties). If $T_0 \preceq T_1$ and $T_1 \preceq T_0$, then T_0 and T_1 are *conformation equivalent.* This means that T_0 and T_1 are functionally interchangeable. There exists a canonical form for trace structures such that two canonical trace structures are conformational equivalent if and only if they are equal.

A trace structure T_1 can also be used to represent a specification. A circuit T_0 satisfies the specification T_1 if T_0 conforms to T_1. Checking that a circuit satisfies a specification would be very expensive if it required checking the failure freedom of compositions with all possible trace structures. However, there exists an operation on trace structures, called *mirroring,* that makes checking conformation practical. If $T = (I, O, S, F)$ is a canonical trace structure, then $T^M = (O, I, S, A^* - P)$ is its mirror. If T' has the same inputs and outputs as T, then the composition of T' and T^M is failure free if and only if

T' conforms T. Thus, checking if a circuit satisfies a specification only requires checking if the composition of the circuit with the mirror of the specification is failure free, which can be done efficiently. If there is an error in the circuit, then usually on a small fraction of the states of the composition need to be explored before a failure is found. When a failure is found, a short trace of the transitions that led to the failure is output to the user.

4 Combining CTL and Trace Theory

Verifying a circuit with both trace theory and CTL involves two steps. The first step is to use trace theory to construct the trace structure representing the composition of the circuit with some environment. The environment may be constructed by computing the mirror of a specification. If the composition has a failure, then this is detected by the verifier during the construction of the trace structure representing the composition, usually after only a small fraction of the structure has been constructed. After detecting a failure, the verifier does a breadth first search to find a shortest failure trace, which is useful for debugging. In this case, no CTL verification is done, and debugging consists of attempting to remove the safety violation detected by the trace theory verifier.

If the composition is failure free, then it is converted to a CTL structure for use by the CTL model checker. This conversion requires that extra information be carried with the trace structures that represent components of the circuit. Trace structures do not explicitly represent the digital values of wires, but CTL structures have this information in the labels of each state. So the verifier adds to trace structures the initial values of the wires of the circuit. The values of the wires in all states is determined using the following construction. The states in the CTL structure that is constructed are each $n+1$ tuples (where n is the number of signals in the circuit): one element of the tuple is the state number of the corresponding state in the automata representing the trace structure of the composition, the other n entries are the values of each of the n signals. The initial state and the next-state relation of the CTL structure are defined in the obvious way. A CTL structure constructed in this way could have many more states, even exponentially more, then the trace structure automata from which it was constructed. Our experiments indicate that in practice, however, the increase in the number of states is usually no more than a factor of 2, and there is often no increase at all.

Correct verification using the model checker depends on having fairness constraints that describe liveness and fairness properties of the circuit components. Ideally information about fairness constraints should be included in the trace structures that model circuit components. When trace structures are converted to CTL structures the fairness constraints would automatically be included. However, the including of fairness constraints in CTL structures is currently done by hand by the user of the verifier.

Once the CTL structure is constructed, the next step is to use the model checker to verify that the circuit satisfies the desired CTL formulas. When a formula is not satisfied, the model checker outputs a counterexample trace, if one exists, to show why the formula is not satisfied.

As an example of verification using both trace theory and CTL, consider the fair mutual exclusion circuit in Figure 1. The signals *ai* and *bi* are request inputs, and *ao* and *bo* are mutually exclusive acknowledge outputs. A 4-phase handshaking protocol is

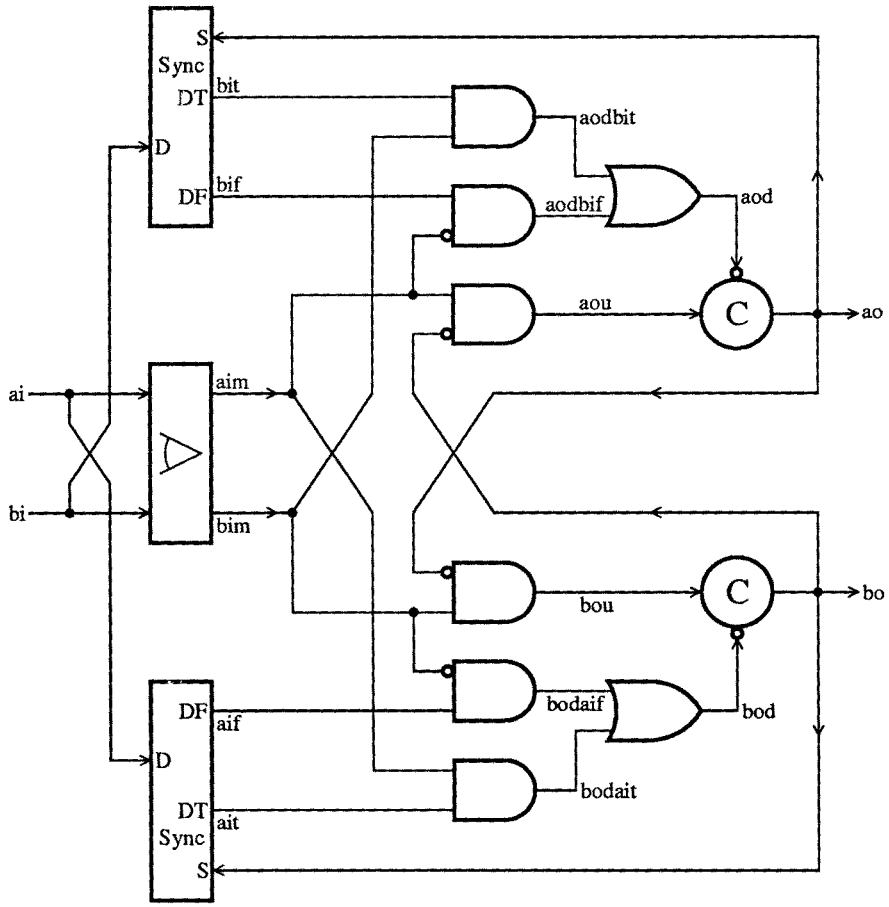

Figure 1: Proposed circuit for fair mutual exclusion.

used, initially all wires are low. The heart of the circuit is a mutual exclusion element that is not assumed to be fair. Also used in the circuit are two synchronizers.

A synchronizer has two inputs, D (data) and S (strobe), and two outputs, DF (data false) and DT (data true). When S goes high, the synchronizer samples the value of D. If D is stable at low voltage, then DF is raised. If D is stable at high voltage, then DT is raised. It is guaranteed that DF and DT will never be high simultaneously. Once either DF or DT is raised, they do not go low until S is taken low, regardless of how D changes. If D changes between when S is raised and either DF or DT is raised, then the synchronizer can go into a metastable state, and either DF or DT may go high. The synchronizer can also go into a metastable state if D changes within a small amount of time (called the *setup time* of the synchronizer) before S is raised. The notion of setup time can not be accurately represented in the speed-independent model. In [9], synchronizers are assumed to have a setup time of zero, which is a rather liberal

assumption. A more conservative alternative is to model a synchronizer so that the output in response to raising S is deterministic only if D has not changed since previous raising of S. Such a synchronizer is said to have a setup time of one cycle. If a circuit is verified correct using a synchronizer with a setup time of one cycle, then that circuit will also be correct if a synchronizer with a setup time of zero is used instead. The converse, however, is not true. We will assume that synchronizers have a setup time of zero when verifying circuits under the speed-independent model. Later, we will address the implications of this assumption.

When the circuit in Figure 1 is verified, a bug is found in less than 40 seconds. (All timing information in this paper is based in running the verifier on a Sun-3/60 with Sun Common Lisp version 2.1.1.) The verifier can output a shortest length trace showing the bug, which can be summarized as follows. The input *bi* remains low throughout the trace. The signals *ai* and *ao* go through one request/acknowledge cycle, this leaves *bif* high. Because we are using the speed-independent model, *bif* may remain high for an arbitrarily long time. Next, *ai* goes high again, which eventually causes *ao* to go high. But *ao* can go high before *bif* returns to a low state, causing a failure in the synchronizer.

The problem is that in the above trace, *ao* is allowed to go high when *bif* might still be high. This can be fixed by replacing the and-gate that drives *aodbif* with a C-element (the inverted input remains inverted). The corresponding change is also made to the and-gate that drives *bodaif*. This way, *bif* must go low before *aod* can go low, which must happen before *ao* can go high.

The error in this circuit could also be found, in theory, using the model checker. However, as currently implemented, this would require constructing the entire CTL structure of the circuit, which our analysis has shown to have at least 400,000 states. This demonstrates how much more effective trace theory can be at finding safety violations quickly. It has been proposed that the model checker be implemented using "lazy" state construction, which might allow it find errors without constructing the full CTL structure. But since the verifier determines the truth of a formula by marking each state of the CTL structure with the truth values of the sub-formulas, there is good reason to believe that even a "lazy" model checker would still construct many more states than the trace theory verifier before finding a bug.

The modification described above does not solve all of the problems with the circuit, but the verifier can be used iteratively to find bugs and output error traces that greatly simplify debugging. The circuit shown in Figure 2 can be derived in this way. Trace theory verifies that this circuit has no safety errors after examining 1332 states in about 50 seconds.

However, it is not possible to check the liveness or fairness properties of this circuit using only trace theory. So we use the system to construct the CTL structure corresponding to the circuit. For this circuit, constructing the CTL structure caused no increase in the number of states needed to model the circuit. Using the model checker, we can verify that the formulas $AG\,AF\,(ai \Leftrightarrow ao)$ and $AG\,AF\,(bi \Leftrightarrow bo)$ are true for the circuit. This means that for every state the circuit might reach, it will eventually reach a state in which *ai* and *ao* have the same value. This implies, among other things, that every request made of the circuit is eventually acknowledged. This liveness property depends on the assumption that whenever a user starts a four-phase handshake, it always completes it and thus releases the mutual exclusion element. This assumption was represented using the fairness constraints $ao \Rightarrow \neg ai$ and $bo \Rightarrow \neg bi$.

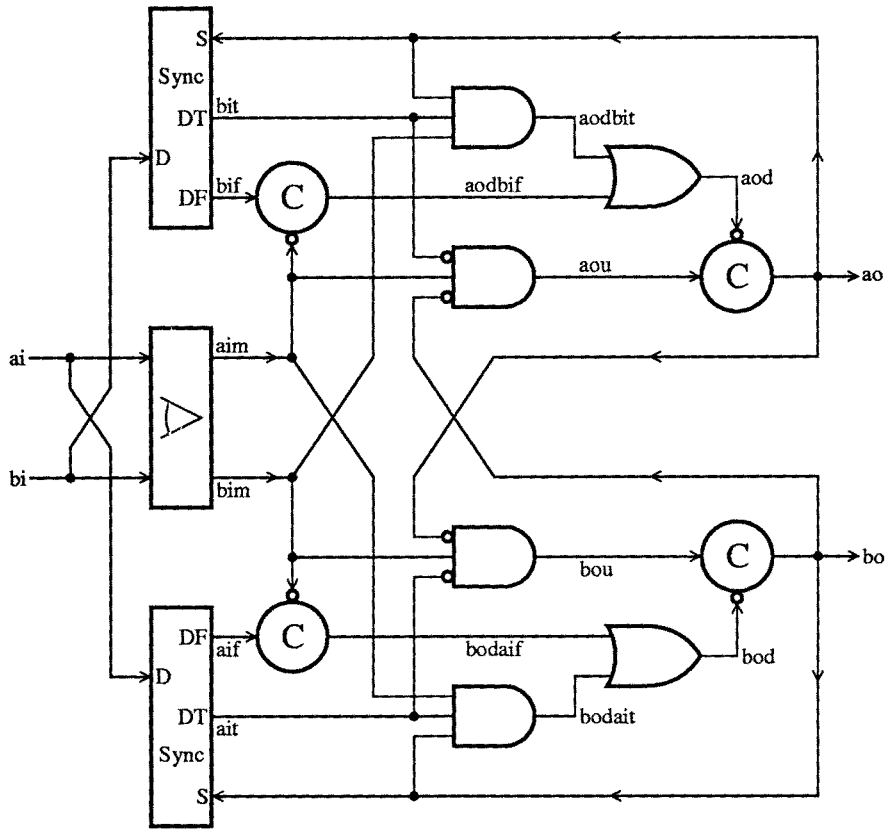

Figure 2: Correct speed-independent fair mutual exclusion circuit.

The above analysis was done using synchronizers with a setup time of zero. If synchronizers with a setup time of one cycle are used instead, the system can verify that there are no safety errors in the circuit, just as before. However, a deadlock is discovered when liveness properties are checked using CTL. One such deadlock is as follows. The circuit first completes a cycle on the b channel, then bi remains low. Then ai goes high. When ao goes high in response, the synchronizer can raise bit. As long as bi remains low, the circuit can make no further progress and is deadlocked. This is an example of the importance of being able to check liveness properties of the circuit. If only safety properties where checked, one might conclude that the circuit is correct for synchronizers with a setup time of one cycle, even though there is a deadlock.

There is a way to solve the deadlock problem described above. A synchronizer with a setup time of zero can be implemented with a synchronizer with a setup time of one cycle by adding some extra circuitry. The details are described in [3]. The key idea is to strobe the synchronizer twice every time the data is to be synchronized, ignoring the synchronizers response to the first strobe. This implementation of a synchronizer with a setup time of zero can be used in the circuit in Figure 2. The result is a speed-

independent fair mutual exclusion that is correct even though it uses synchronizers with a setup time of one cycle.

There has been debate concerning whether it is possible to build a speed-independent or delay-insensitive fair mutual exclusion circuit [1,9,12]. In [9], Martin attempted to show that such a circuit existed by constructing one. Unfortunately, Martin later discovered an error in his circuit [8]. The trace theory verifier has also found errors in this circuit. The circuit presented in Figure 2, which has been verified correct, provides an example of a speed-independent fair mutual exclusion circuit.

5 Timing models

In the verification methods we've discussed so far, only speed-independent circuits can be represented. We wish to extend trace theory to allow the representation of a large class of timing models for different circuit components. We call the extended theory *trace theory with discrete time.* The extension allows traces to contain additional symbols not corresponding to physical wires. The presence of such a symbol in a trace represents the passage of some amount of time. More formally, we allow trace structures of the form (I, O, V, S, F), where V is a set of symbols (disjoint from I and O), and where S and F are sets of traces over the alphabet $A = I \cup O \cup V$. The elements of V do not correspond to any physical wires, so they are called *virtual wire names.* In order to give an intuitive explanation of the intended meaning of such structures, we will only consider the case in which V contains a single element, call it φ.

The presence of a φ in a trace indicates the passage of a unit of time, call it τ. The trace $\varphi\varphi x\varphi$ represents a single behavior in which a transition occurs on wire x at time $T_0 + 2\tau$, where T_0 is the time at which the behavior described by the trace began. Traces in trace theory with discrete time can also be interpreted as approximating continuous time. Interpreted in this way the above trace represents the set of behaviors in which a transition occurs on wire x at a time t such that $T_0 + 2\tau \leq t < T_0 + 3\tau$.

Figure 3 is an automata describing the S set of a buffer with input b, output x, and virtual wire φ. Notice that since S is prefix closed, all of the states in this automata are accepting states. The F set of this buffer is equal to $(SI - S)A^*$. Interpreted in discrete time, this buffer clearly has a minimum delay of 2τ and a maximum delay of 3τ. We say such a buffer obeys a discrete 3/2 rule.

In continuous time, the 3/2 rule states that the minimum delay of each component is 2 units and the maximum delay is 3 units. The buffer in Figure 3 approximates a buffer that obeys the 3/2 rule in continuous time. To see this, consider the set of continuous time behaviors equal to the union of the sets of behaviors represented by each trace in the P set of the buffer in Figure 3. Any behavior a continuous 3/2 rule buffer might display is contained in this set.

We have also developed a version of trace theory called *trace theory with continuous time* in which every transition in a trace has a real number associated with it to represent the exact time at which the transition took place. By proving relationships between the two kinds of trace theory, we can increase our confidence that trace theory with discrete time accurately models physical systems. It can be shown for a large class of specifications and component models that if a circuit conforms to a specification when modeled with trace theory with discrete time, then it is also conforms to the specification in trace theory with continuous time. If the specifications and component models are restricted

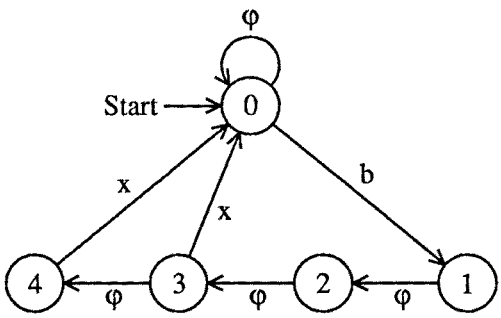

Figure 3: Automata that accepts the S set of a 3/2 rule buffer.

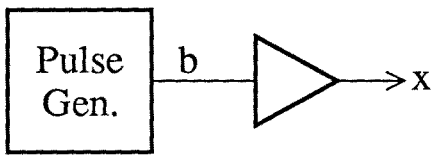

Figure 4: Example circuit for demonstrating effects of timing assumptions.

somewhat more, then the converse is also true. Contained in this set is a large class of specifications that include timing properties such as requiring a circuit to respond to an input within a certain amount of time. As an example of why the converse is not true in general, consider two buffers in series with input b and output x, and three buffers in series with input c and output y. If b transitions before c, then by the continuous 3/2 rule, x will transition before y. In the discrete 3/2 rule, x is guaranteed to precede y if and only if b transitions before c and there is a φ between the b and c transitions. This illustrates that there can be extra behaviors in trace theory with discrete time that are not in trace theory with continuous time. These extra behaviors can cause a circuit to not conform to a specification in trace theory with discrete time.

Although the discrete 3/2 rule is not as strong as the continuous 3/2 rule, it does place useful restrictions on the order of transitions in a circuit. As an example of this, we consider a circuit formed using the output of a pulse generator as the input to a buffer as in Figure 4. The pulse generator has a period of 8τ and a 50% duty cycle. Its S set is described by the automata in Figure 5 and its F set is empty. The buffer is the same as that represented in Figure 3.

The trace structure representing the resulting circuit has an empty F set and its S set is given by the automata in Figure 6. Since the maximum delay of the buffer is 3τ and the pulse generator waits 4τ between outputting transitions, there are never two consecutive b transitions without an x transition in between. This fact could not be represented with the speed-independent timing model.

Timing models other than the 3/2 rule can also be represented. It is clear how to modify the buffer in Figure 3 to have a minimum delay of $m\tau$ and a maximum delay of $n\tau$ for any non-negative integers m and n such that $m \leq n$. The buffer can also be made

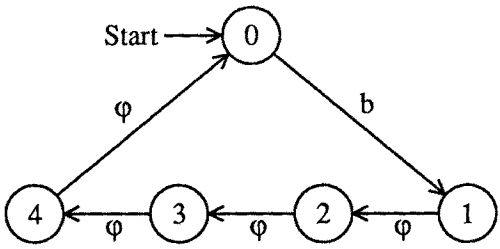

Figure 5: Automata that accepts the S set of a pulse generator.

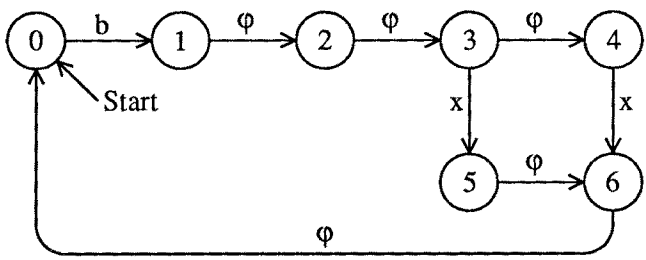

Figure 6: Automata representing the behavior of the example circuit in Figure 4.

to have an unbounded maximum delay by removing state 4 and adding a transition on φ from state 3 back to state 3.

The trace theory verifier has been extended to include virtual wires in this way. This is combined with CTL as before. If a composition is failure free then the resulting finite automata, the behaviors of which reflect any timing assumptions made, is converted to a CTL structure.

As an example of verifying circuits using timing models, consider again the circuit in Figure 1. We will analyze this circuit using the discrete 3/2 timing model. We model each of the gates to have a minimum delay of 2τ and a maximum delay of 3τ. The outputs of the synchronizers and the mutual exclusion element have a minimum delay of 2τ. When these two components are in a metastable state, we cannot bound their delay; otherwise their maximum delay is 3τ. The mutual exclusion element may be in a metastable state when both of its inputs are high and both its outputs are low; otherwise, it is guaranteed not to be metastable. The synchronizer may go into a metastable state if the D input changes to soon before the strobe input S goes high, or after the strobe goes high and before any output goes high. We define a setup time of 3τ for the synchronizers: if the D input is stable for 3τ before the strobe goes high, and remains stable until an output goes high, then the synchronizer is assumed to not go into a metastable state.

Under these timing assumptions, trace theory verifies the circuit in about 200 seconds after examining 3826 states. The fairness of the circuit is verified using CTL as before. Thus, this circuit, which appeared to be incorrect in our previous analysis, is in fact

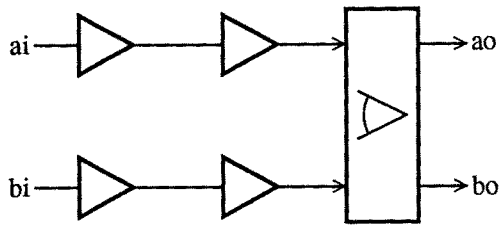

Figure 7: Simplified circuit for fair mutual exclusion under the discrete 3/2 rule.

correct given reasonable timing assumptions. This circuit is smaller and simpler than the circuit we showed to be correct in the speed-independent model.

We can do even better than this. The circuit in Figure 1 does not make use of the maximum delay of the arbiter in states that are not metastable. Rather than an elaborate circuit using synchronizers to insure fairness, we design a circuit that limits the rate at which a user can reissue requests. Then the mutual exclusion element is guaranteed to respond to a request from one user before receiving the second of two requests from another user. This results in the rather trivial circuit in Figure 7. The verifier quickly shows that this circuit satisfies all the necessary safety and liveness properties. The delays insure that no user can make requests quickly enough to cause unfair behavior, even if the mutual exclusion element is unfair. The circuit is faster and much simpler than the previous circuits. If there is a known lower bound on the delay from when the circuit takes an acknowledge low to when the corresponding user takes its request high, then some or all of the delays in the circuit in Figure 7 can be safely removed, resulting in an even faster circuit.

The previous example shows that given reasonable assumptions on the relative delays of components, designing a fair mutual exclusion circuit is almost trivial. This is not a surprising result, the problem of speed-independent fair mutual exclusion is often perceived to be of only theoretical interest. However, to the author's knowledge, this is the first time this result has been demonstrated by an automatic verifier. The ability of this verifier to accurately model this kind of phenomenon increases its usefulness as a design tool.

6 Conclusions and Future Research

We have presented a verification system that combines the advantages of CTL and trace theory. The resulting system often finds bugs after exploring only a small fraction of the states of a circuit, greatly simplifies checking for hazards, and allows the verification of liveness and fairness properties.

We have also shown how this verifier has been extended to use more liberal timing models than the speed-independent model. Our example indicates how this can aid in the design and verification of circuits that are smaller and faster than similar speed-independent circuits. It is also possible to verify circuits against specifications that include timing properties such as requiring a circuit to respond to an input within a

certain amount of time.

Although the verifier presented here uses time linear in the number of states in the circuit, the time used can be exponential in the number of components of the circuit. Though we see no way to improve the asymptotic worst case performance of the verifier, we believe its performance on typical circuits can be improved. One way to do this uses the fact that in many physically plausible timing models the order of two physical wire names in a trace is irrelevant when there is no virtual wire name between them. This fact can be used to reduce the number of states explored during the verification of a circuit using such a timing model.

There is nothing about the method presented here that restricts its application to asynchronous circuits, it could also be applied to other concurrent systems. Using this method to study examples of other kinds of concurrent systems is an area for future research.

Acknowledgements

Ed Clarke has been my advisor during this work, which could not have been completed without his guidance. David Dill provided much help and encouragement. My many discussions with Alain Martin about disciplined methods for designing asynchronous circuits were necessary preparation for designing the fair mutual exclusion circuits in this paper. I also had helpful discussions with David Long.

References

[1] David L. Black. On the existence of fair delay-insensitive arbiters: trace theory and its limitations. *Distributed Computing*, 1(4):205–225, 1986.

[2] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12):1035–1044, 1986.

[3] Jerry R. Burch. The design of a delay-insensitive fair mutual exclusion circuit. In Preparation.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[5] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* PhD thesis, Department of Computer Science, Carnegie Mellon University, 1988.

[6] David L. Dill. Trace theory for automatic hierarchical verification of speed-independent circuits. In Jonathan Allen and F. Thomson Leighton, editor, *Advanced Research in VLSI: Proceedings of the Fifth MIT Conference*, MIT Press, 1988.

[7] David L. Dill and Edmund M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, Part E 133(5), 1986.

[8] Alain J. Martin. Personal Communication.

[9] Alain J. Martin. *A Delay-Insensitive Fair Arbiter.* Technical Report 5193:TR:85, California Institute of Technology, Computer Science Department, 1985.

[10] Alain J. Martin. A synthesis method for self-timed vlsi circuits. In *Proceedings of the IEEE International Conference on Computer Design*, 1987.

[11] Robin Milner. *Lecture Notes in Computer Science. Volume 92: A Calculus of Communicating Systems.* Springer-Verlag, 1980.

[12] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits and systems. *Distributed Computing*, 1(4):197–204, 1986.