Verifying Liveness Properties By Verifying Safety Properties (Extended Abstract)

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

Abstract

Conventional techniques for automatically verifying liveness properties of circuits involve explicitly modeling infinite behaviors with either infinite paths through a Kripke structure or with strings in an ω-regular language. This paper describes how *timed trace structures* [2, 3] can be used to convert liveness properties (including unbounded liveness properties such as strong fairness) to safety properties. Such properties can then be modeled and verified using only finite traces. No new algorithms are needed. All that is required is a new interpretation of what behaviors are represented by the finite traces. A mapping is defined between timed trace structures and *complete trace structures* [5], which contain infinite traces, to show that this new interpretation makes sense. The method is demonstrated on a fair mutual exclusion circuit.

1 Introduction

The primary goal of research in automated circuit verification is to devise algorithms that check whether circuits enjoy certain desirable properties. It is common to classify these properties into *safety* and *liveness* properties. Informally, safety properties state that nothing bad happens, while liveness properties state that something good happens.

In formalisms that explicitly model time (such as timed trace structures, described below), it is also possible to express *bounded liveness* properties. Bounded liveness properties state that something good happens within a bounded amount of time. This is in contrast to *unbounded liveness* properties, which state that something good happens eventually without specifying a time bound. A bounded liveness property can be viewed as a form of safety property; conventional methods used to automatically verify safety properties are also adequate for bounded liveness properties.

There are many systems described in the literature that can automatically verify unbounded liveness properties. All of these explicitly model behaviors as being infinite: most often as infinite paths through a Kripke structure or as strings in an ω-regular language. This paper describes a method for converting unbounded liveness properties to bounded liveness properties, and thereby to safety properties. This makes it possible to verify unbounded liveness properties (such as strong fairness) without explicitly modeling infinite behaviors. Instead, methods based on prefix-closed trace structures of finite traces are shown to be adequate. This result adds new insight concerning Black's proof [1] that delay-insensitive fair mutual exclusion cannot be represented using conventional finite traces.

The method merely requires a new interpretation of the behaviors represented by timed trace structures. It was not necessary to develop new algorithms or modify the automatic verifier in any way. The method is demonstrated using a simple buffer circuit as an example. In addition, a mutual exclusion circuit is used to show how strong fairness properties can be modeled and verified.

Using this method a mapping can be defined from timed trace structures on finite traces to complete traces structures [5] on infinite traces. Conformation between trace structures is preserved under this mapping. Thus, the mapping can be used to check whether a specification properly captures the desired liveness properties.

2 Timed Trace Theory

We can only give a short overview of timed trace theory in the space of an extended abstract; the interested reader may refer to [2] and [3]. The formalism is based on Dill's trace theory [5, 6]. We begin by describing trace theory, and then show how it can be extended to include timing information.

In trace theory based verification digital circuits and their specifications are modeled by *trace structures,* which are ordered 4-tuples of the form $\mathcal{T} = (I, O, S, F)$. The set I is the set of input wire names of the circuit; O is the set of output wire names. The set $A = I \cup O$ is called the *alphabet* of $\mathcal{T}$. 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.

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 [8], 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.

As an example, consider how a buffer might be modeled by a trace structure $\mathcal{T} = (I, O, S, F)$. 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 shows that liveness properties cannot be modeled with this form of trace theory.

A composition operation (denoted by "$\circ$") can be defined on trace structures. Let $\mathcal{T}_0 = (I_0, O_0, S_0, F_0)$ and $\mathcal{T}_1 = (I_1, O_1, S_1, F_1)$ be trace structures. The composition of $\mathcal{T}_0$ and $\mathcal{T}_1$ 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 $A_0 = I_0 \cup O_0$ and $A_1 = I_1 \cup O_1$. If x is a trace in $(A_0 \cup A_1)^*$, 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

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

The operation **hide** is also defined on trace structures. The trace structure denoted by **hide**$(D)[(I,O,S,F)]$ (where D is a subset of O) is equal to $(I, O-D, S\,|_{A'}, F\,|_{A'})$, where $A' = I \cup (O - D)$ and the projection operation is extended to sets of traces in the normal manner.

A trace structure is said to be *failure free* when its 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. The S and F sets of trace structures are represented in the verifier with deterministic finite automata. Checking if the composition of two trace structures is failure free is done in time linear in the product of the sizes of these automata. If a composition is not failure free, then usually only a small fraction of the states of the composition need to be explored before a short error trace can be given.

Let $\mathcal{T}_0$ and $\mathcal{T}_1$ be trace structures. Then, $\mathcal{T}_0$ *conforms to* $\mathcal{T}_1$ (written $\mathcal{T}_0 \preceq \mathcal{T}_1$) if $I_0 = I_1$ and $O_0 = O_1$ and for any trace structure $\mathcal{T}$, the composition of $\mathcal{T}$ and $\mathcal{T}_1$ being failure free implies that the composition of $\mathcal{T}$ and $\mathcal{T}_0$ is failure free. The idea captured here is that if a circuit works correctly with $\mathcal{T}_1$ as a component, then it works correctly with $\mathcal{T}_1$ replaced by $\mathcal{T}_0$ (up to safety properties). If $\mathcal{T}_0 \preceq \mathcal{T}_1$ and $\mathcal{T}_1 \preceq \mathcal{T}_0$, then $\mathcal{T}_0$ and $\mathcal{T}_1$ are *conformation equivalent.* This means that $\mathcal{T}_0$ and $\mathcal{T}_1$ 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 $\mathcal{T}_1$ can also be used to represent a specification. A circuit $\mathcal{T}_0$ satisfies the specification $\mathcal{T}_1$ if $\mathcal{T}_0$ conforms to $\mathcal{T}_1$. 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 $\mathcal{T} = (I,O,S,F)$ is a canonical trace structure, then $\mathcal{T}^M = (O, I, S, A^* - (S \cup F))$ is its mirror. If $\mathcal{T}'$ has the

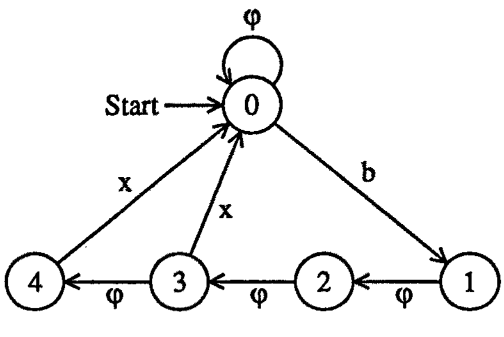

Figure 1: Automata that accepts the S set of a 3/2 rule buffer.

same inputs and outputs as $\mathcal{T}$, then the composition of $\mathcal{T}'$ and $\mathcal{T}^M$ is failure free if and only if $\mathcal{T}'$ conforms $\mathcal{T}$. 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. This is done in time linear in the product of the sizes of the automata representing the circuit and the specification. If there is an error in the circuit, then usually on a small fraction of the state space needs 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.

2.1 Adding Timing Information

In standard trace theory, only speed-independent circuits can be represented. However, trace theory can be extended to allow the representation of a large class of timing models for different circuit components. We call the extended theory *timed trace theory*. The extension allows traces to contain additional symbols not corresponding to physical wires. The presence of such a symbol in a trace is interpreted as representing the passage of some fixed amount of time. (Later in the paper, we describe an alternative interpretation that allows for the modeling of unbounded liveness properties.) 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 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.

Figure 1 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.

Consider a circuit formed using the output of a pulse generator as the input to a buffer as in Figure 2. The pulse generator has a period of 8τ and a 50% duty cycle. Its S set is described by the automata in Figure 3 and its F set is empty. The buffer is the same as

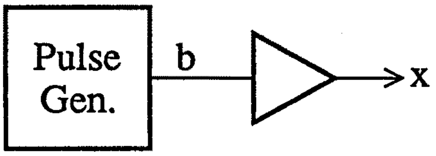

Figure 2: Example circuit for demonstrating effects of timing assumptions.

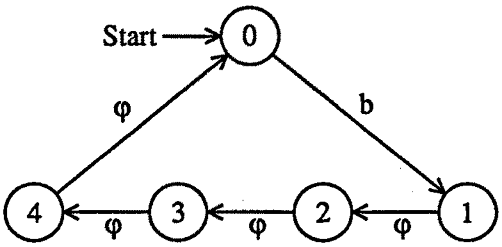

Figure 3: Automata that accepts the S set of a pulse generator.

that represented in Figure 1. The trace structure representing the resulting circuit has an empty F set and its S set is given by the automata in Figure 4. 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.

It is clear how to modify the buffer in Figure 1 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 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. Checking that a circuit satisfies a specification has the same complexity as before. Examples of using the verifier to check the correctness of circuits can be found in [2] and [3].

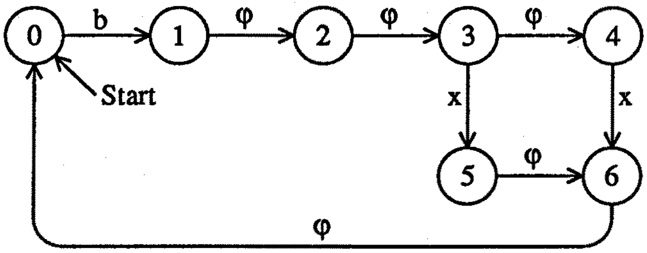

Figure 4: Automata representing the behavior of the example circuit in Figure 2.

3 Converting Liveness to Safety

Let trace structure $\mathcal{T}$ model a buffer with input b, output x, and virtual wire φ_s that has a minimum delay of zero and a maximum delay of one. Normally, the presence of a φ_s in a trace in $\mathcal{T}$ indicates the passage of a unit of time. For the purpose of converting liveness properties to safety properties, we propose a different interpretation. A transition on b can be seen as placing an obligation on the buffer to eventually toggle x in response. We interpret the nth φ_s in a trace to mean that the buffer has met all such obligations that the environment placed on it before the $(n-1)$th φ_s in the trace. In this interpretation, there need not be a constant, or even bounded, amount of time between each φ_s.

As an example of how this interpretation can be used to verify liveness properties, consider verifying that two buffers in series conform to a single buffer. Let trace structure $\mathcal{T}_1$ model a buffer with input b and output c, and let $\mathcal{T}_2$ model a buffer with input c and output x. Both trace structures have a single virtual wire φ_i, with a minimum delay of zero and a maximum delay of one. In order to show that $\mathcal{T}_1 \circ \mathcal{T}_2$ conforms to the specification, we compose $\mathcal{T}_1 \circ \mathcal{T}_2$ with a constraining trace structure $\mathcal{C}$. The trace structure $\mathcal{C}$ has no inputs or outputs, but has virtual wires φ_i and φ_s. The sole purpose of $\mathcal{C}$ is to constrain the transitions on the virtual wires so that there must be at least three transitions of φ_i between any two transitions of φ_s. Three is the minimum necessary to assure that the two buffers in series satisfy their obligation to transition x before a transition of φ_s. The automatic verifier can be used to show that

$$\text{hide}(\{c, \varphi_i\})[\mathcal{T}_1 \circ \mathcal{T}_2 \circ \mathcal{C}] \preceq \mathcal{T}.$$

If $\mathcal{C}$ is chosen badly, then the verifier may incorrectly report that the circuit does not conform to the specification. The user is required to find a $\mathcal{C}$ that allows the verification to go through. Regardless of what $\mathcal{C}$ is used, however, the verifier will never report that a circuit conforms to a specification when in fact it does not. In this sense, the verification method is *conservative.*

4 Mapping to Complete Traces

In the previous section, we described a specification for a live buffer in terms of a timed trace structure. But it may not be intuitively clear that the timed trace structure captures the properties of a live buffer. This section describes a way of addressing that problem.

In [5], Dill describes a version of trace theory, called complete trace theory, where liveness properties are modeled using infinite traces. Dill describes a complete trace structure that is a specification for a live buffer, as a special case of a live gate. It is easy to give an intuitive argument that this complete trace structure accurately captures the properties of a live buffer. If it could be shown that the timed trace specification for a live buffer was in some way equivalent to the complete trace specification, that would be strong evidence that the timed trace specification is correct.

One way to do this is to define a mapping Φ from timed trace structures to complete trace structures such that

$$\mathcal{T}_1 \preceq \mathcal{T}_2 \Rightarrow \Phi(\mathcal{T}_1) \preceq \Phi(\mathcal{T}_2),$$

and such that the timed trace specification of the live buffer is mapped by Φ to the complete trace specification. As an example of why the implication is in only one direction, let $\mathcal{T}_1$ be a buffer with a maximum delay of one and let $\mathcal{T}_2$ be a buffer with a maximum delay of two. Both $\mathcal{T}_1$ and $\mathcal{T}_2$ have a minimum delay of zero, and have one virtual wire φ. In this case $\Phi(\mathcal{T}_1) = \Phi(\mathcal{T}_2)$, so $\Phi(\mathcal{T}_2) \preceq \Phi(\mathcal{T}_1)$. However $\mathcal{T}_2$ clearly does not conform to $\mathcal{T}_1$.

The definition of such a mapping Φ, and proofs of its properties, are included in the full version of the paper [4].

5 Fair Mutual Exclusion Example

The verification example above for buffers only deals with a simple kind of liveness property. In this section to use the verifier to check a strong fairness property, which is a more difficult kind of liveness property.

The specification for a fair mutual exclusion element has inputs ai and bi and outputs ao and bo. It requires that any transition on ai be followed by a transition on ao with a maximum delay of one, as measured against the virtual wire φ_s. A similar requirement is placed on bi and bo. Again, this does not represent a bounded response time. The response time is unbounded since there is not assumed to be a bound on the amount of time between transitions of φ_s. However, this specification can not be satisfied by any implementation unless it is assumed that the environment will always eventually release the mutual exclusion element. This assumption is represented by introducing another virtual wire φ_r. After a request is granted by the mutual exclusion element, the environment must release it with a maximum delay of one, as measured against φ_r. Otherwise, it is a failure. This does not actually bound the time for which a channel can hold the mutual exclusion element. It simply means that the environment must constrain φ_r so that the mutual exclusion element is released before there are to many φ_r transitions. Let $\mathcal{T}$ be the trace structure representing this specification. Also, let $\mathcal{T}_1$ represent a buffer with input ai and output ao and a maximum delay of one, as measured against φ_s. Finally, let $\mathcal{T}_2$ represent an inverter with input bo and output bi and a maximum delay of one, as measured against φ_r. The automatic verifier can be used to show that

$$\mathbf{hide}(\{bi, bo, \varphi_r\})[\mathcal{T} \circ \mathcal{T}_2] \preceq \mathcal{T}_1.$$

This is evidence that the formal specification captures the desired informal specification for a fair mutual exclusion element.

The verifier was used to verify the speed-independent fair mutual exclusion circuit described in [2]. This involves modeling each component of the circuit as having a maximum delay of one relative to a virtual wire φ_i, and then constraining φ_s relative φ_i and φ_r.

It is straightforward to define a complete trace structure that is a specification for fair mutual exclusion. The mapping Φ described above maps the timed trace theory specification for fair mutual exclusion to such a complete trace specification. This shows that the notion of fair mutual exclusion is accurately modeled by the timed trace theory specification.

In [1] Black shows that conventional finite traces cannot model delay-insensitive fair mutual exclusion. His arguments can be used to show that conventional finite traces cannot model speed-independent fair mutual exclusion if there is no bound on the number requests granted to one channel before a request is granted to the other channel, as in the specification above. Black proposed that infinite traces be used to model fair mutual exclusion. We have shown how trace theory can be modified to model fair mutual exclusion without introducing infinite traces.

6 Conclusions

We have shown how liveness properties, such as unbounded fairness, can be converted to safety properties. This makes possible the automatic verification of liveness properties using finite traces. No new algorithms are needed, just a reinterpretation of the meaning of timed traces. We showed this new interpretation makes sense by defining a mapping Φ from timed trace structure to complete trace structures that have infinite traces. This mapping preserves conformation, and maps specifications in timed trace theory to specifications in complete trace theory that are more obviously correct.

Unfortunately, tests indicate that the method presented here may not be efficient in practice. The same fair mutual exclusion circuit that was verified in this paper was verified more quickly using CTL in [2]. Verifying the circuit with timed traces required that a much larger state space be explored than was necessary using CTL. Thus, the results in this paper may be of more theoretical than practical interest. However, there may be areas of practical application. For example, in machine aided verification techniques like those in [9], a circuit is proved correct by proving it satisfies an invariant. Previously, only safety properties could be verified in this way. But the method described here might be adapted to allow verification techniques based on invariants to also verify unbounded liveness properties.

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] Jerry R. Burch. Combining CTL, trace theory and timing models. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1989.

[3] Jerry R. Burch. Modeling timing assumptions with trace theory. In *IEEE International Conference on Computer Design*, October 1989.

[4] Jerry R. Burch. Verifying liveness properties by verifying safety properties. In Robert Kurshan and Edmund M. Clarke, editors, *Computer-Aided Verification, Proceedings of the 1990 Workshop*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1991. To appear.

[5] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1988. Also appeared as [7].

[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. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1989.

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

[9] Jorgen Staunstrup, Stephen J. Garland, and John V. Guttag. Localized verification of circuit descriptions. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France*, volume 407 of *Lecture Notes in Computer Science.* Springer-Verlag, June 1989.