REAL-TIME STATE MACHINES AND CIRCUIT VERIFICATION WITH MODAL FUNCTIONS

Victor Yodaiken
George S. Avrunin[1]

CMPSCI Technical Report 93–04
January 1993

Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003

[1]Research partially supported by grants from the National Science Foundation and the Office of Naval Research

Real-time state machines and circuit verification with modal functions

Victor Yodaiken
Department of Computer Science
University of Massachusetts at Amherst
Amherst, MA 01003
yodaiken@cs.umass.edu

George S. Avrunin*
Department of Mathematics and Statistics
University of Massachusetts at Amherst
Amherst, MA 01003
avrunin@math.umass.edu

Abstract

Recent work on automated verification of circuits has highlighted the problems of concise representation, composition, and manipulation of state machines with large state sets. In this paper, we show how state machines that capture both the real-time behavior and connective structure of circuits can be defined, parameterized, composed, and refined, using modal functions [4, 5]. Our work is currently focused on "by hand" methods of verification as a step towards automated methods.

1 Introduction

The behavior of circuits with up to a few hundred states can be modeled, with some difficulty, using the traditional methods of state diagrams and transition tables. Automation, via model checking and simulation, can extend the applicability of state machine models to circuits of millions of states. And recent work has shown that symbolic representations of state machines as binary decision diagrams can extend model checking to certain circuits with far larger state sets [2]. In this paper, we show how the state machine approach can be extended again, to provide detailed models of the real-time behavior and design structure of still more complex circuits. Our work makes use of modal functions [4, 5], a class of state dependent functions that are essentially Moore machines (state transducers), presented in a concise form amenable to abstraction, parameterization, and composition. We show how modal functions can be used to provide practical descriptions of both individual circuit elements such as gates, at a level of detail adequate to represent features including setup and propagation times and response to glitches, and circuits composed of many such elements. We also show how the *encapsulation* operator for modal functions leads to a natural composition operation for circuits that corresponds closely to circuit

*Research partially supported by grants from the Office of Naval Research and the National Science Foundation.

diagrams. Our current work [1] is chiefly concerned with methods for "by-hand" verification of circuits as a step towards automated verification, and we give a small example illustrating some of our proof methods here.

If $\mathtt{f}$ is a modal function, the value of $\mathtt{f}(x)$ may depend on both x and on the *current state* of $\mathtt{f}$. The state variable of a modal function is never made explicit. Instead, we specify the state dependent behavior of modal functions in terms of the effect of inputs representing environmental signals, commands, and physical events. For example, a modal function $\mathtt{Alarm}(t)$, might be defined so that $\mathtt{Alarm}(t) = \mathit{true}$ if and only if at least t inputs representing the passage of a unit of physical (real) time have been accepted since the most recent input representing a reset command. We write $\mathtt{f}:_A X \rightarrow Y$ to define $\mathtt{f}$ as a modal function from X to Y with state sensitive to inputs from alphabet A. That is, $\mathtt{f}(x)$ depends on the sequence of inputs from A that has driven $\mathtt{f}$ from its initial state. If $\mathtt{f}$ represents a circuit, the elements of A will represent *samples* of the signal levels on the input pins of the circuit, X will name the output pins of the circuit, and Y will consist of the signal values that the circuit may generate on its output pins.

Imagine that we have access to a perfectly accurate digital oscilloscope or logic analyzer that may be set to an arbitrary sample rate and sensitivity. We may set the sample rate to some fixed time interval, say a femto-second, or to correspond to some abstract unit, perhaps depending on a fundamental clock rate of a design or on a property of the technology used to fabricate the circuit. The measuring device could sample voltage to an arbitrary (finite) precision, and might sample such factors as current flow or temperature as well. If we attach our imaginary measuring device to a circuit, we will obtain a picture of circuit behavior in terms of real-time samples of input and output signal levels. Since the measuring device is perfectly accurate, n samples will correspond to exactly n units of physical time[1]. We may then view the circuit as a "black-box", a modal function that accepts real-time signal samples as inputs and that maps output pins to real-time signal samples in each state. In this model, any circuit element that has state is a state machine (modal function): wires that impose delays are state machines, inverters are state machines, and pass transistors are state machines. To represent composite circuits, we can connect our "black-box" models of individual circuits by associating each input pin of each component circuit with either an output pin of one of the component circuits or an input wire of the composite circuit. When an input is applied to the state machine of the composite circuit, the wiring diagram is used to produce an appropriate input sample, in parallel, for the state machine of each component. There is a composition operator on modal functions that can be made to naturally correspond to this kind of circuit connection.

In this paper, modal functions and operators on modal functions are always distinguished from ordinary functions by typography: $\mathtt{f}$ is modal, f is not, (START) is a modal function operator, $\sum$ is an ordinary function operator. Otherwise, we have attempted to use as little nonstandard notation as possible. We have also presented modal functions in an informal fashion, stressing intuition over technical detail. Modal functions can be

[1]We do not assume the existence of a perfectly accurate clock device. The input sample stream is a discretization of the physical input signals, not a representation of the output of a clock device.

shown to be equivalent to Moore machines, in that each modal function captures the input-output relationship of a Moore machine, every Moore machine is captured by some modal function, and the parallel composition operation used to connect modal functions corresponds to a type of automata product called the feedback product. The development of this correspondence is outside the scope of this paper, but may be found in [5].

Section 2 introduces modal functions, a simple, unit delay model of circuits, and methods for composition. Section 3 develops a more sophisticated timing model, incorporating setup and propagation delays. A brief conclusion discusses prospects for automating and current work on the verification of asynchronous circuits.

2 Modal functions and real-time circuits.

Let C be a circuit with input pins $I = \{i_1, ..., i_n\}$ and output pins $O = \{o_1, ..., o_m\}$. As real time passes, C senses signal levels on the input pins and generates signals on the output pins. It may be the case that several logical pins correspond to a single physical pin: for example, when the circuit technology allows for wired-ors or when a pin is bidirectional. Or it may be that a pin name p is an element of both I and O, but refers to distinct physical pins. For example the input and output pins of a delay element may be given identical names. We specify the behavior of C by defining a modal function $\mathtt{C}{:}_S O \to V$ where V is set of signal levels, and at each moment of time the value of $\mathtt{C}(o)$ is the signal that our hypothetical measuring device would measure on output pin o. The inputs to modal functions representing circuits are ordinary functions that assign a signal level to each input pin. For instance the inputs to a latch will assign signal levels to input pins *data-in* and *clock*. In this paper, we will only allow for two signal levels, 0 and 1, so the input alphabet of a circuit with input pins I will be the set of all maps $I \to \{0,1\}$. Thus, a modal function representing a circuit with input pins I and output pins O will be a state dependent map $\mathtt{C}{:}_{\{I\to\{0,1\}\}} O \to \{0,1\}$. We often abbreviate this as $\mathtt{C}{:}_I O \to \{0,1\}$.

2.1 Specifying modal functions

A modal function $\mathtt{f}$ can be defined from ordinary functions g and h, by setting the initial state value of $\mathtt{f}(x)$ to be equal to $g(x)$, and setting the value of $\mathtt{f}(x)$ in the state *after* input a to be equal to $h(a,x,y)$ where y is the current state value of $\mathtt{f}(x)$. For example, suppose the input alphabet consists of input samples $a\colon \{\textit{data-in}, \textit{enable}\} \to \{0,1\}$ and define a unit delay latch device so that $\mathtt{UnitLatch}(\textit{data-out}) = 0$ in the initial state, and after an input a, if $a(\textit{enable}) = 1$, then $\mathtt{UnitLatch}(\textit{data-out}) = a(\textit{data-in})$, but if $a(\textit{enable}) = 0$ then $\mathtt{UnitLatch}(\textit{data-out})$ will be unchanged. Here $h(a,x,y) \overset{\text{def}}{=} y$ **if** $a(\textit{enable}) = 0$ **else** $a(\textit{data-in})$. This type of definition provides a recursive algorithm for calculating $\mathtt{f}(x)$ from the history: the sequence of inputs that have driven $\mathtt{f}$ from its initial state. In practice, the history sequence is always left implicit and we reason about the state dependent properties of modal functions in the abstract. Often, we don't even completely specify the modal function because the circuit we want to describe is not completely specified. For example, we may not specify the behavior of a modal function $\mathtt{ClockedDevice}(p)$ if the clock signals are too rapid or too irregular.

The state of a modal function may be modified by two function operators: $(\text{START})\mathtt{f}(x)$

resets $\texttt{f}$ to its initial state before applying $\texttt{f}$ to x, and $(\text{AFTER } a)\texttt{f}(x)$ advances $\texttt{f}$ from its current state by input a before applying $\texttt{f}$ to x. The formal definition of the simple latch of the previous paragraph is as follows.

$$(\text{START})\texttt{UnitLatch}(\mathit{data}) \stackrel{\text{def}}{=} 0$$

$$(\text{AFTER } a)\texttt{UnitLatch}(\mathit{data}) \stackrel{\text{def}}{=} \begin{cases} a(\mathit{data}) & \text{if } a(\mathit{enable}) = 1; \\ \texttt{UnitLatch}(\mathit{data}) & \text{otherwise.} \end{cases}$$

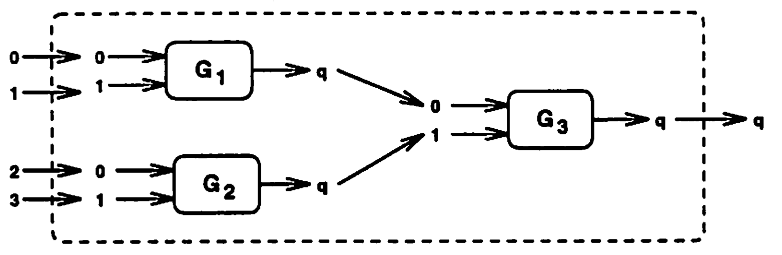
Note that the left hand side of the next state definition refers to the value of the function in the next state, and the right hand side only refers to the value in the current state. If $\texttt{h}$ is a previously defined modal function, then $(\text{AFTER } a)\texttt{f}(x) \stackrel{\text{def}}{=} \texttt{h}(a, x, \texttt{f}(x))$ defines behavior of $\texttt{f}$ in the next state to depend on both the behavior of $\texttt{f}$ in the current state, and behavior of $\texttt{h}$ in the current state. Here, we assume that $\texttt{h}$ and $\texttt{f}$ share the same history. We do not insist that definitions of modal functions follow this format exactly: for instance $\texttt{f}$ may be a function on more than one argument and $\texttt{h}$ might reference the current state value of $\texttt{f}(r(x))$ or might ignore the input symbol a. We do insist that the right hand side of a next state definition only refer to the $\texttt{f}(x)$ in the current state — the definition $(\text{AFTER } a)\texttt{f}(x) \stackrel{\text{def}}{=} h(x, (\text{AFTER } a)\texttt{f}(x))$ is circular.

For many purposes, circuit elements with a single unit of delay provide a sufficiently detailed model. This is especially true for clocked circuits where we can assume that data wires are synchronized properly with the clock signal. In the next section, we discuss a more sophisticated notion of timing, but a few examples showing the simpler timing model may be of interest.

A unit delay element can be defined so that $I = O$, $(\text{START})\texttt{UDelay}(i) \stackrel{\text{def}}{=} 0$, and $(\text{AFTER } a)\texttt{UDelay}(i) = a(i)$. Thus, the value of $\texttt{UDelay}(i)$ in the current state is equal to the signal level that was most recently sampled on input pin i. A unit delay inverter is given by $\texttt{UnitInvert}(i) \stackrel{\text{def}}{=} 1 - \texttt{UDelay}(i)$. A unit delay and-gate with output pin q can be defined by $(\text{START})\texttt{UnitAnd}(q) \stackrel{\text{def}}{=} 0$ and $(\text{AFTER } a)\texttt{UnitAnd}(q) \stackrel{\text{def}}{=} \prod_i a(i)$. A unit-delay or-gate is specified by replacing multiplication with boolean addition.

We assume that there is a single input history sequence, so if $\texttt{C}$ and $\texttt{C}'$ both reference input pin i, then both will sense identical signals at the same moment. This common reference to an input pin does not necessarily correspond to a physically shared pin. For example, $\texttt{UnitAnd}(q) = \prod_i \texttt{UDelay}(i)$ because $\texttt{UDelay}(i)$ captures the most recent signal applied to pin i, and $\texttt{UnitAnd}(q)$, is the boolean product of those signals. We will often make use of $\texttt{UDelay}(i)$ to refer to the signal that was most recently sampled on the input pin i of an unrelated circuit. Using $\texttt{UDelay}$, we can easily specify a negative edge triggered latch: a latch that is enabled when it detects a transition from 1 to 0 on its enable input pin. A negative edge triggered latch will latch new data only when $a(\mathit{enable})$ is 0 and the enable line is currently set to 1. This is true only when $a(\mathit{enable}) < \texttt{UDelay}(\mathit{enable})$.

$$(\text{AFTER } a)\texttt{UnitEdgeL}(\mathit{data\text{-}out}) \stackrel{\text{def}}{=} \begin{cases} a(\mathit{data\text{-}in}) & \text{if } a(\mathit{enable}) < \texttt{UDelay}(\mathit{enable}); \\ \texttt{UnitEdgeL}(\mathit{data\text{-}out}) & \text{otherwise.} \end{cases}$$



$$\omega \overset{\text{def}}{=} \begin{cases} q \to \mathtt{G}_3(q) \\ (\mathtt{G}_1, 0) \to 0 \\ (\mathtt{G}_1, 1) \to 1 \\ (\mathtt{G}_2, 0) \to 2 \\ (\mathtt{G}_2, 1) \to 3 \\ (\mathtt{G}_3, 0) \to \mathtt{G}_1(q) \\ (\mathtt{G}_3, 1) \to \mathtt{G}_2(q) \end{cases}$$

Figure 1: An and-gate with 4 inputs pins composed of 3 and-gates each having 2 input pins

2.2 Composition of circuit elements

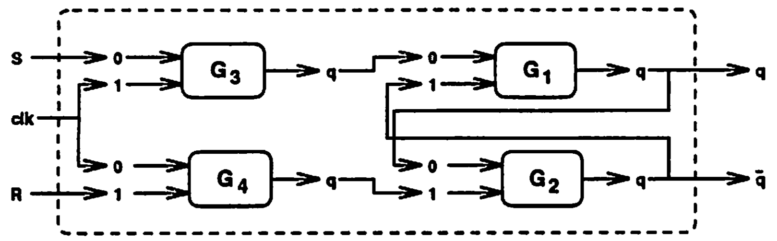
In this section, we describe a limited form of the parallel composition operator of modal functions that has been tailored to allow natural specification of interconnected circuits.

If we have n circuit modal functions $\mathtt{C}_1, ..., \mathtt{C}_n$, we can construct a composite modal function $\mathtt{C}$ by connecting these functions according to a wiring diagram. The wiring diagram associates each input pin of each subcircuit with a single source: either an input pin of the composite circuit, or an output of a connected subcircuit. The wiring diagram also associates each output pin of the composite circuit with a source — the source must be an output pin of some subcircuit. Wiring diagrams may be considered as total (non-modal) functions from destination pins to source pins — each destination pin must have exactly one source pin. It is convenient to treat a wiring diagram as a set of associations *dest-pin* $\to$ *source-pin.* If input pin i of subcircuit $\mathtt{C}_j$ is driven by input pin i' of the composite circuit, then $(\mathtt{C}_j, i) \to i'$ will be in the wiring diagram. If input pin i of subcircuit $\mathtt{C}_j$ is driven by output pin o of subcircuit $\mathtt{C}_k$, then $(\mathtt{C}_j, i) \to \mathtt{C}_k(o)$ will be in the wiring diagram. If output pin o of the circuit is driven by the output pin o' of subcircuit $\mathtt{C}_j$, then $o \to \mathtt{C}_j(o')$ will be in the diagram. We write $\mathtt{C}(o) \overset{\text{def}}{=} \text{CONNECT}((\mathtt{C}_1, ..., \mathtt{C}_n), \omega)(o)$ to define the composite circuit constructed by connecting together copies of $\mathtt{C}_1, ..., \mathtt{C}_n$ so that their inputs are controlled by wiring diagram ω.

Consider how we might construct an and-gate with four input pins from 3 and-gates that only have two input pins. That is, suppose that we have a specification $\mathtt{UnitAnd}_{\{0,1\}}\{q\} \to \{0,1\}$ and we want to specify a gate $\mathtt{BigAnd}_{\{0,1,2,3\}}\{q\} \to \{0,1\}$ that is constructed by connecting three copies of UnitAnd.

Let $\mathtt{G}_j(q) \overset{\text{def}}{=} \mathtt{UnitAnd}(q)$ for $j = 1,2,3$. The inputs to the composite circuit will be maps $a\colon \{0,1,2,3\} \to \{0,1\}$. We want to encapsulate the 2 input and-gates within BigAnd so that each encapsulated gate receives the correct input, as determined by the wiring. To define BigAnd we first define a wiring map that associates each input pin of the component gates and the output pin of the composite circuit with a source. The source will either be the output of one of the component gates, or an input wire, as shown in the more conventional wiring diagram on the left in Figure 1.

Now set $\mathtt{BigAnd}(q) \overset{\text{def}}{=} \text{CONNECT}((\mathtt{G}_1, \mathtt{G}_2, \mathtt{G}_3), \omega)(q)$, where ω is as shown on the right in Figure 1. Since there is a single unit delay at each stage, the output will be the and of the second most recent input sample. An expression $(\text{AFTER } a_1)(\text{AFTER } a_2)...(\text{AFTER } a_n)\mathtt{f}(x)$



$$\sigma \overset{\text{def}}{=} \begin{cases} q \rightarrow \mathtt{G}_1(q) \\ \bar{q} \rightarrow \mathtt{G}_2(q) \\ (\mathtt{G}_1, 0) \rightarrow \mathtt{G}_3(q) \\ (\mathtt{G}_1, 1) \rightarrow \mathtt{G}_2(q) \\ (\mathtt{G}_2, 0) \rightarrow \mathtt{G}_1(q) \\ (\mathtt{G}_2, 1) \rightarrow \mathtt{G}_4(q) \\ (\mathtt{G}_3, 0) \rightarrow S \\ (\mathtt{G}_3, 1) \rightarrow clk \\ (\mathtt{G}_4, 0) \rightarrow R \\ (\mathtt{G}_4, 1) \rightarrow clk \end{cases}$$

Figure 2: A clocked S-R flip-flop constructed from nand-gates

is evaluated from left to right: first apply input a_1, then input a_2, and so on. Thus $(\mathtt{AFTER}\ a_1)(\mathtt{AFTER}\ a_2)\mathtt{BigAnd}(q) = \prod_i a_1(i)$.

An n unit delay constructed from unit delay elements can be defined similarly. Set $\mathtt{D}_j(i) \overset{\text{def}}{=} \mathtt{UDelay}(i)$ for $j = 1, ..., n$, and define wiring diagram γ as follows:

$$\gamma \overset{\text{def}}{=} \begin{cases} i \rightarrow \mathtt{D}_n(i) \\ (\mathtt{D}_1, i) \rightarrow i \\ (\mathtt{D}_{j+1}, i) \rightarrow \mathtt{D}_j(i) \text{ for } 0 < j < n \end{cases}$$

Now define $\mathtt{UDelay}^n(i) \overset{\text{def}}{=} \mathtt{CONNECT}((\mathtt{D}_1, ..., \mathtt{D}_n), \gamma)(i)$.

In the next section, a 2 input nand-gate with setup time s and propagation time r will be specified. The input pins are named 0 and 1 and the output pin is named q in keeping with the specification of a unit delay gate. A clocked S-R flip-flop has data input pins S and R, a clock input pin clk, and output pins q and $\bar{q}$ (see, for example, [3, page 209]). To define a clocked S-R flip-flop, we set $\mathtt{F}(q) \overset{\text{def}}{=} \mathtt{CONNECT}((\mathtt{G}_1, \mathtt{G}_2, \mathtt{G}_3, \mathtt{G}_4), \sigma)$ where each $\mathtt{G}_j$ meets the specification of a nand-gate and σ is the wiring map shown in Figure 2.

In the next section, we will also give a higher level specification of what this device is supposed to do. Here, however, we can specify a shift register constructed from flip-flops.

Suppose that $\mathtt{F}_1, ..., \mathtt{F}_n$ represent flip-flops constructed as above. We construct a shift register with input pins S, R and clk, by connecting the input clock pin and R pin of each flip-flop to the clk and R pins of the shift register, connecting pin S of $\mathtt{F}_1$ to pin S of the shift register, and connecting pin S of each $\mathtt{F}_{j+1}$ to output pin q of $\mathtt{F}_j$. There will be a shift register output pin q_j connected to the output pin q of each flip-flop $\mathtt{F}_j$. Define the circuit diagram α as follows.

$$\alpha \overset{\text{def}}{=} \begin{cases} q_j \rightarrow \mathtt{F}_j(q) \\ (\mathtt{F}_1, S) \rightarrow S \\ (\mathtt{F}_j, R) \rightarrow R \text{ for } 0 < j \leq n \\ (\mathtt{F}_j, clk) \rightarrow clk \text{ for } 0 < j \leq n \\ (\mathtt{F}_{j+1}, S) \rightarrow \mathtt{F}_j(q) \text{ for } 0 < j < n \end{cases}$$

And let $\texttt{Shift}(q_j) \overset{\text{def}}{=} \text{CONNECT}((\text{F}_1, ..., F_n), \alpha)(q_j)$.

To prove properties of composite modal functions, it is often necessary to examine the behavior of internal elements. In order to access internal information in a composite circuit, we *splice* in modal functions that measure or test some property of an internal circuit element or its history. $(\text{SPLICE } \texttt{f.f}_j)\text{g}(x)$ inserts the modal function $\text{g}(x)$ into a composite function $\texttt{f}$, so that the input stream for component $\texttt{f}_j$ is copied into the input stream of g. More precisely, if $\texttt{f}(x) \overset{\text{def}}{=} \text{CONNECT}((\texttt{f}_1, ..., \texttt{f}_n), \omega)$, then $(\text{SPLICE } \texttt{f.f}_j)\text{g}(x)$ is the modal function $\text{CONNECT}((\texttt{f}_1, ..., \texttt{f}_n, \text{g}), \omega')$ where ω' is identical to ω except that ω' arranges for g to get the same inputs as $\texttt{f}_j$, and makes the outputs of the composite function be the outputs of g. For example, $(\text{SPLICE } \texttt{BigAnd.G}_3)\texttt{UDelay}(i)$ is the modal function $\text{CONNECT}((\text{G}_1, ..., G_3, \texttt{UDelay}), \omega')(i)$ where

$$\omega' \overset{\text{def}}{=} \begin{cases} i \to \texttt{UDelay}(i) \\ \texttt{UDelay}(0) \to \text{G}_1(q) \\ \texttt{UDelay}(1) \to \text{G}_2(q) \\ (\text{G}_1, 0) \to 0 \\ (\text{G}_1, 1) \to 1 \\ (\text{G}_2, 0) \to 2 \\ (\text{G}_2, 1) \to 3 \\ (\text{G}_3, 0) \to \text{G}_1(q) \\ (\text{G}_3, 1) \to \text{G}_2(q). \end{cases}$$

Thus, $\texttt{BigAnd}(q) = \prod_i(\text{SPLICE } \texttt{BigAnd.G}_3)\texttt{UDelay}(i)$. In general, if $(\texttt{f}_j, i) \to \texttt{f}_k(o) \in \omega$, then $(\text{AFTER } a)(\text{SPLICE } \texttt{f}.f_j)\texttt{UDelay}(i) = \texttt{f.f}_k(o)$ And if $(\texttt{f}_j, i) \to i' \in \omega$, then $(\text{AFTER } a)(\text{SPLICE } \texttt{f}.f_j)\texttt{UDelay}(i) = a(i')$.

3 More sophisticated models of timing

We begin this section by defining some operators on modal functions that tell us how long a signal has been stable, high, or low. Then we define modal functions that tell us whether a signal must be set to a certain value within some time interval, or must stay at a certain value for some time interval. These operators allow for quite sophisticated specification of circuit timing properties.

Define $(\text{HIGH})\texttt{f}(x)$ as follows.

$$(\text{START})(\text{HIGH})\texttt{f}(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } \texttt{f}(x) > 0; \\ 0 & \text{otherwise} \end{cases}$$

$$(\text{AFTER } a)(\text{HIGH})\texttt{f}(x) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } (\text{AFTER } a)\texttt{f}(x) < 1; \\ 1 + (\text{HIGH})\texttt{f}(x) & \text{otherwise.} \end{cases}$$

Note that this definition is not circular because we refer to the next value of $\texttt{f}(x)$, not the next value of $(\text{HIGH})\texttt{f}(x)$, on the right hand side and, presumably, $\texttt{f}$ is already defined.

A similar operator counts how long a modal function has been low.

$$(\text{START})(\text{LOW})\mathtt{f}(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } \mathtt{f}(x) < 1; \\ 0 & \text{otherwise} \end{cases}$$

$$(\text{AFTER } a)(\text{LOW})\mathtt{f}(x) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } (\text{AFTER } a)\mathtt{f}(x) > 0; \\ 1 + (\text{LOW})\mathtt{f}(x) & \text{otherwise.} \end{cases}$$

Finally, define (STABLE)$\mathtt{f}(x)$ to be the time that has passed since $\mathtt{f}(x)$ last changed. This is never less than 1.

$$(\text{START})(\text{STABLE})\mathtt{f}(x) \overset{\text{def}}{=} 1$$

$$(\text{AFTER } a)(\text{STABLE})\mathtt{f}(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } (\text{AFTER } a)\mathtt{f}(x) \neq \mathtt{f}(x); \\ 1 + (\text{STABLE})\mathtt{f}(x) & \text{otherwise.} \end{cases}$$

To illustrate, suppose that some input pin p is being used as a clock pin to synchronize input signals on a set of input data pins D. The environment should only change the signal on pin p when the signal on every pin $d \in D$ has been stable for at least t time units. When synchronization fails, the environment must hold p and each element of D low for at least t' time units to resynchronize. Define $\mathtt{Sync}(t, t', p, D)$ so that $(\text{START})\mathtt{Sync}(t, t', p, D) \overset{\text{def}}{=} 0$ and

$$(\text{AFTER } a)\mathtt{Sync}(t, t', p, D)$$
$$\overset{\text{def}}{=} \begin{cases} 0 & \text{if } a(p) \neq \mathtt{UDelay}(p) \\ & \text{ and } (\exists d \in D)(\text{STABLE})\mathtt{UDelay}(d) < t \\ & \\ 1 & \text{if } (\text{AFTER } a)(\text{LOW})\mathtt{UDelay}(p) \geq t \\ & \text{and } (\forall d \in D)(\text{AFTER } a)(\text{LOW})\mathtt{UDelay}(d) \geq t \\ & \\ \mathtt{Sync}(t, t', p, D) & \text{otherwise.} \end{cases}$$

Note that once the timing constraint is violated $\mathtt{Sync}(t, t', p, D)$ stays at 0 until resynchronization takes place.

Next, we need to be able to specify the future behavior of a signal. If u is a sequence of inputs $u = \langle a_1, ..., a_n \rangle$, then $(\text{AFTER } u)\mathtt{f}(x)$ advances the state of $\mathtt{f}$ by a_1, then by a_2, and so on up to a_n before evaluating $\mathtt{f}(x)$. Let $(\text{AFTER } \langle\rangle)\mathtt{f}(x) = \mathtt{f}(x)$ and let $(\text{AFTER } \langle a_1, ..., a_n \rangle)\mathtt{f}(x) = (\text{AFTER } \langle a_1, ..., a_{n-1} \rangle)\ (\text{AFTER } a_n)\mathtt{f}(x)$. This extension to AFTER is all we need to define operators that test what will happen in the future.

Define $(\text{WITHIN } n)\mathtt{f}(x)$ to be true (1) if and only if $\mathtt{f}(x)$ must become non-zero within n time units.

$$(\text{WITHIN } n)\mathtt{f}(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if, for all } u \in A^n, \\ & \text{there exists a prefix } z \text{ of } u \\ & \text{so that } (\text{AFTER } z)\mathtt{f}(x) \neq 0, \\ 0 & \text{otherwise} \end{cases}$$

An unrestricted form of WITHIN tests to see if $\mathtt{f}(x)$ must become non-zero after some number of time units.

$$(\text{EVENTUALLY})\mathtt{f}(x) \stackrel{\text{def}}{=} (\exists n)(\text{WITHIN } n)\mathtt{f}(x)$$

Finally, $(\text{FOR } n)\mathtt{f}(x)$ is true if and only if $\mathtt{f}(x)$ is non-zero and will stay non-zero for at least $n-1$ more steps.

$$(\text{FOR } n)\mathtt{f}(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if, for all } u \in A^{n-1}\text{, for every prefix } z \text{ of } u \ (\text{AFTER } z)\mathtt{f}(x) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

As an illustration, consider an improved specification of a logic gate. Here we will specify a nand-gate so that a modal function G is a nand-gate with setup time s, propagation time r, and switching time h if and only if: whenever any input pin has been low for s time units, within r time units the output pin will be set to 1 and stay at 1 for at least h time units, and whenever all the input pins have been high for at s time units, within r time units the output pin will be set to 0 and will stay at 0 for at least h time units. Otherwise, the behavior of the gate is unspecified.

$$\begin{aligned} &\mathtt{NandGate}(\mathtt{G}, I, O, s, r, h) \\ \leftrightarrow\ &(\exists i)(\text{LOW})\mathtt{UDelay}(i) \geq s \rightarrow (\text{WITHIN } r)(\text{FOR } h)\mathtt{G}(q) = 1 \\ &\wedge(\forall i)(\text{HIGH})\mathtt{UDelay}(i) \geq s \rightarrow (\text{WITHIN } r)(\text{FOR } h)\mathtt{G}(q) = 0 \end{aligned}$$

In many cases we can let $r = h$, the time for a new signal to propagate to the output limits the switching time of the gate. To illustrate proof methods, let's prove the theorem that if $h = r$ $(\forall i)(\text{HIGH})\mathtt{UDelay}(i) \geq s + r$ implies that $(\text{FOR } r)(\mathtt{G}(q) = 0)$. This theorem is a result of the following general observation.

Proposition 1. *Let* P *and* Q *be modal functions taking values in* $\{0,1\}$. *Suppose that* $(\text{START})\mathtt{P} = 0$ *and* $(\text{HIGH})\mathtt{P} \geq x$ *implies* $(\text{WITHIN } y)(\text{FOR } y)\mathtt{Q}$ *in every state. Then* $(\text{HIGH})\mathtt{P} \geq x + j$ *implies* $(\text{WITHIN } y - j)(\text{FOR } y)\mathtt{Q}$, *for all* $j \leq y$.

Proof. We prove the proposition by induction on j. The case $j = 0$ is trivial, so we may assume the result holds in every state for $j \leq n$. We wish to prove that it holds for $j = n + 1$.

To establish this, we argue by induction on state. The result holds vacuously for $j = n+1$ in the initial state, since $(\text{START})\mathtt{P} = 0$. We assume it holds in the current state, and show that it holds in the next state. Suppose then that $(\text{AFTER } a)(\text{HIGH})\mathtt{P} \geq x+n+1$. Then $(High)\mathtt{P} \geq x + n$, and we conclude that $(\text{WITHIN } y - n)(\text{FOR } y)\mathtt{Q}$. If $(\text{FOR } y)\mathtt{Q}$ does not hold in the current state, then we must have$(\text{AFTER } a)(\text{WITHIN } y - n - 1)(\text{FOR } y)\mathtt{Q}$, and we are done. We can therefore assume that $(\text{FOR } y)\mathtt{Q}$ holds.

In this case, we have $(\text{AFTER } a)(\text{FOR } y-1)\mathtt{Q}$. Furthermore, since we are assuming that $(\text{AFTER } a)(\text{HIGH})\mathtt{P} \geq x+n+1 \geq x+n$, we know by induction that $(\text{AFTER } a)(\text{WITHIN } y - n)(\text{FOR } y)\mathtt{Q}$. As before, we consider the two ways this can hold. If $(\text{AFTER } a)(\text{FOR } y)\mathtt{Q}$ is true, we are done. So we can assume that $(\text{AFTER } a)(\text{FOR } y)\mathtt{Q}$ does not hold, in which case we must have $(\text{AFTER } a)(\text{AFTER } a')(\text{WITHIN } y - n - 1)(\text{FOR } y)\mathtt{Q}$ for all inputs a'.

Combining this with the fact that $(\textsc{after}\ a)(\textsc{for}\ y-1)\texttt{Q}$, we see that, after the input a, $\texttt{Q}$ must be 1 for $y-1$ time units and, within $y-n-1$ time units, it must take the value 1 and hold it for the following $y-1$ units. Since $n \geq 0$, this implies that $(\textsc{after}\ a)(\textsc{within}\ y-n-1)(\textsc{for}\ y)\texttt{Q}$, completing the induction step. □

Now, we can provide a detailed model of the real-time behavior of clocked flip-flop with three input pins S (set), R (reset) and clk (clock), and two output pins q and $\bar{q}$. The specification takes the form of an implication: *if the inputs signals are properly synchronized, then the flip-flop output will behave correctly.* If the flip-flop timing requirements are violated, the behavior of the flip-flop is unspecified. Timing is given in terms of four parameters: setup time s, propagation time r, hold time h, and resynchronize time y. When the clock signal on pin clk is high, the signals on S and R should have been stable for at least s time units, and when the clock signal is raised, it should stay high for at least h time units. Furthermore, when the clock signal is high, it should never be the case the the signals on R and S are both high. If the timing requirements are violated, the flip-flop can be resynchronized by holding the clock signal high and R and S at complementary values for for at least y time units. If the flip-flop inputs are synchronized, then the flip-flop will latch a value when the clock signal is dropped, will propagate this value to output within r time units, and will not otherwise change its output value. We start by defining a function that tests whether the input signals meet these synchronization requirements.

$$(\textsc{start})\texttt{SyncFF}(s,h,y) \overset{\text{def}}{=} 0$$

$$(\textsc{after}\ a)\texttt{SyncFF}(s,h,y) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } a(clk) > 0 \\ & \quad \vee[(\textsc{after}\ a)(\textsc{stable})\texttt{Udelay}(S) < s \\ & \quad \wedge(\textsc{after}\ a)(\textsc{stable})\texttt{Udelay}(R) < s] \\ & \text{or } a(clk) = 0 \\ & \quad \wedge 0 < (\textsc{high})\texttt{UDelay}(clk) < h \\ 1 & \text{if } (\textsc{after}\ a)(\textsc{low})\texttt{UDelay}(clk) > y \\ & \quad \wedge(\textsc{after}\ a)(\textsc{stable})\texttt{UDelay}(R) > y \\ & \quad \wedge(\textsc{after}\ a)(\textsc{stable})\texttt{UDelay}(S) > y \\ & \quad \wedge a(S) \neq a(R) \\ \texttt{SyncFF}(s,h,y) & \text{otherwise.} \end{cases}$$

To specify the flip-flop, we also need a modal function that will latch the input data that will be propagated to the output of the flip-flop.

$$(\text{AFTER } a)\mathtt{SyncFF}(s,h,y) \rightarrow$$

$$(\text{AFTER } a)\mathtt{L}(s,h,y) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } a(R) > a(S); \\ 1 & \text{if } a(R) < a(S); \\ \mathtt{L}(s,h,y) & \text{otherwise.} \end{cases}$$

The flip-flop itself is now quite simple. A modal function F is a flip-flop with timing parameters s, h, y, and r if and only if, in any state, within r time units either $\mathtt{F}(q)$ will be equal to the current value latched by L or the environment will violate the timing specifications.

$$\mathtt{IsFlipFlop}(\mathtt{F},s,h,y,r) \leftrightarrow$$
$$\mathtt{L}(s,h,y) = x \rightarrow (\text{WITHIN } r)(\neg\mathtt{SyncFF}(s,h,r) \vee \mathtt{F}(q) = x = 1 - \mathtt{F}(\bar{q}))$$

For appropriate choices of timing parameters, we can prove that the composite flip-flop circuit of the previous section does implement a flip-flop. If the timing parameters are chosen poorly, the composite circuit will not work, as in real-life. There is not space to completely develop the proof here, but a quick sketch should convey the flavor. First, we note that (SPLICE $\mathtt{F.G_3}$)UDelay(0) will be identical to UDelay(S) and (SPLICE $\mathtt{F.G_3}$)UDelay(1) will be identical to UDelay(clk). Similarly, the clock and R signals are passed directly to $\mathtt{G_4}$. It follows that if the flip-flop is resynchronized and the gates are specified to be fast enough, at the moment of resynchronization, the outputs of both gates will have been complementary and stable for some time that we can determine using the nand-gate theorem proved above. Now, we note that (SPLICE $\mathtt{F.G_3}$)(HIGH)$\mathtt{G_3}(q) = n$ with $n > 0$ implies that (SPLICE $\mathtt{F.G_1}$)(HIGH)UDelay(0) $= n - 1$ because of the wiring diagram. That is, if the output of gate $\mathtt{G_3}$ changes, in the next time unit input pin 0 of $\mathtt{G_1}$ will change. So, it becomes an easy matter to show that resynchronization can reset the outputs of gates $\mathtt{G_1}$ and $\mathtt{G_2}$. The rest of the proof is accomplished by showing that the outputs of $\mathtt{G_3}$ and $\mathtt{G_4}$ determine the value of L and that their values propagate to $\mathtt{G_1}$ and $\mathtt{G_2}$.

4 Conclusion

Digital circuit designers and computer architects have traditionally used state machines to provide an intuitive and faithful model of the behavior of discrete circuits. But, traditional methods for working with state machines are too limited for a detailed examination of real-time signal propagation, parameterized specifications, or circuits constructed by connecting subcircuits. In this paper, we have sketched a modal function based approach to circuit state machines that permits the concise definition of state machines with immense state sets, the use of parameters, and convenient composition. We have shown how modal functions may be used to give compact representations of the large state machines required to represent circuit elements in realistic detail and have introduced a variant on modal function composition that is tailored to the requirements of digital circuit connection.

Although we are currently applying these methods in the manual verification of some families of circuits [1], we hope to eventually be able to automate a substantial part of the process. We are able to prove theorems describing the properties of large circuits, including their timing, in terms of properties of smaller subcircuits from which they are composed. In at least some cases, the properties of the components could be checked by existing automated methods, thereby verifying properties of the larger composite circuits. We are beginning to explore the integration of modal function specifications with the descriptions used by existing model checking tools. Another approach would be to develop an automated theorem prover suitable for use with modal functions. Most of our proofs are based on an induction on state, and the Boyer-Moore theorem prover might provide a good basis for this effort. Finally, the modal functions can be regarded as symbolic representations of state diagrams, and it may be possible to develop methods for manipulating them directly, along the lines of the BDD-based model checking methods.

References

[1] G. S. Avrunin and V. Yodaiken. Compositional analysis of some unclocked circuits. In preparation.

[2] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages*, pages 343–354, Jan. 1992.

[3] J. Millman. *Micro-electronics.* McGraw-Hill, 1979.

[4] V. Yodaiken. The algebraic feedback product of automata. A state machine based model of concurrent systems. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verfication '90*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 591–614. American Mathematical Society, 1990. An expanded version of this paper is available by anonymous ftp from dime.cs.umass.edu.

[5] V. Yodaiken. Modal functions for concise representation of finite automata. *Information Processing Letters*, Nov 20 1991.