Formal Verification of Digital Circuits Using Symbolic Ternary System Models*

Randal E. Bryant
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Carl-Johan H. Seger
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z2 Canada

Abstract

Ternary system modeling involves extending the traditional set of binary values $\{0, 1\}$ with a third value X indicating an unknown or indeterminate condition. By making this extension, we can model a wider range of circuit phenomena. We can also efficiently verify sequential circuits in which the effect of a given operation depends on only a subset of the total system state.

This paper presents a formal methodology for verifying synchronous digital circuits using a ternary system model. The desired behavior of the circuit is expressed as assertions in a notation using a combination of Boolean expressions and temporal logic operators. An assertion is verified by translating it into a sequence of patterns and checks for a ternary symbolic simulator. The methodology has been used to verify a number of full scale designs.

1 Introduction

Most formal models for hardware verification assume that every signal always has a well-defined, discrete value. For example, a *binary* model assumes that each signal must be either 0 or 1. In this paper we present a methodology for formal verification in which a third value X is added to the set of possible signal values, indicating an unknown or indeterminate logic value. By shifting to a ternary system model, we gain several advantages.

As a first advantage, this extension makes it possible to model an increased range of circuit phenomena. For example, we can deal with circuits in which nondigital voltages are generated in the course of normal circuit operation. This occurs frequently when modeling circuits at the switch-level [4], due to (generally transient) short circuits or charge sharing. We can also deal with circuits in which indeterminate behavior occurs due either to timing hazards or to circuit oscillation.

*This research was supported by the Defense Advanced Research Projects Agency, ARPA Order Number 4976, and by the National Science Foundation, under grant number MIP-8913667.

In all of these cases, the modeling algorithm expresses this uncertainty by assigning value X to the offending circuit nodes, indicating that the actual digital value cannot be determined [6, 7].

As a second advantage, we can efficiently verify many aspects of digital circuit behavior by representing the circuit with a ternary system model. We do this by *ternary symbolic simulation*, in which a simulation algorithm designed to operate on scalar values 0, 1, and X, is extended to operate on a set of symbolic values. Each symbolic value indicates the value of a signal for many different operating conditions, parameterized in terms of a set of symbolic Boolean variables. Since the value X indicates that a signal could be either 0 or 1 (or a non-digital voltage), we can often represent many different operating conditions by the constant value X, rather than with a more complex symbolic value.

Simulators that support ternary modeling intentionally err on the side of pessimism for the sake of efficiency. That is, they will sometimes produce a value X even where exhaustive case analysis would indicate that the value should be binary (i.e., 0 or 1). On the other hand, symbolic simulation avoids this pessimism, because it can resolve the interdependencies among signal values. By combining the expressive power of symbolic values with the computational efficiency of ternary values, we can trade off precision for ease of computation.

In earlier work, we demonstrated the utility of ternary modeling for verifying a variety of circuits [1, 5]. This earlier work demonstrated the viability of circuit verification by symbolic simulation, but it fell short in terms of generality, ease of use, and degree of automation. In this paper, we correct this shortcoming by presenting a formal state transition model for a ternary system, a formal syntax for expressing desired properties of the system, and an algorithm to decide whether or not the system obeys the specified property. Our state transition system is quite general, and is compatible with a number of circuit modeling techniques. The specifications take the form of *symbolic trajectory formulas* mixing Boolean expressions and the temporal *next-time* operator. Finally, our decision algorithm is based on ternary symbolic simulation. It tests the validity of an assertion of the form $[A \Longrightarrow C]$, where both A and C are trajectory formulas. That is, it determines whether or not every state sequence satisfying A (the "antecedent") must also satisfy C (the "consequent"). It does this by generating a symbolic simulation sequence corresponding to the antecedent, and testing whether the resulting symbolic state sequence satisfies the consequent.

An important property of our algorithm is that it requires a comparatively small amount of simulation and symbolic manipulation to verify an assertion. The restrictions we impose on the formula syntax guarantee that there is a unique weakest symbolic sequence satisfying the antecedent. Furthermore, the symbolic

manipulations involve only variables explicitly mentioned in the assertion. Unlike other symbolic circuit verifiers [2], we do not need to introduce extra variables denoting the initial circuit state or possible primary inputs. Finally, the length of the simulation sequence depends only on the depth of nesting of temporal next-time operators in the assertion.

By modifying the COSMOS symbolic switch-level simulator[3], we have been able to implement the algorithm described in this paper and to verify several full scale circuit designs. The following table indicates the performance of our prototype verifier on several different circuits. All CPU times were measured on a DEC 3100 (a 10–20 MIPS machine). We also list the maximum memory requirement of the process, as this is more often the limiting factor in symbolic manipulation than is CPU time.

Circuit	Transistors	CPU Time	Memory
64 × 32 bit moving data stack	16,470	1.25 min.	3.1 MByte
64 × 32 bit stationary data stack	15,873	7.5 min.	5.7 MByte
1K static RAM	6,875	3.7 min.	9.5 MByte

2 Ternary System

Let $\mathcal{B} = \{0, 1\}$ be the set of the binary values and let $\mathcal{T} = \{0, 1, X\}$. The value X is introduced to denote an "unknown", or "don't care" value.

Define the partial order $\sqsubseteq$ on $\mathcal{T}$ as follows: $a \sqsubseteq a$ for all $a \in \mathcal{T}$, $X \sqsubseteq 0$, and $X \sqsubseteq 1$. The partial ordering orders values by their "information content." That is, X indicates an absence of information while 0 and 1 represent specific, fully-defined values.

We say that ternary values a and b are *compatible*, denoted $a \sim b$, when there is some value $c \in \mathcal{T}$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$. Also, given two compatible ternary values a and b, the *join* between them, denoted $a \sqcup b$, is defined to be the smallest element $c \in \mathcal{T}$ in the partial order such that $a \sqsubseteq c$ and $b \sqsubseteq c$.

It is convenient to define an algebra over $\mathcal{T}$ with operators $\sqcup$, $\cdot_t$, $+_t$, and $^{-t}$, where the latter are the obvious extensions of the corresponding Boolean operations $\cdot$ (product), $+$ (sum), and $^-$ (complement).

Let $\mathcal{T}^n$, $n \geq 1$, denote the set of all possible vectors of ternary values of length n, i.e., $\{\langle a_1, \ldots, a_n \rangle | a_i \in \mathcal{T}, 1 \leq i \leq n\}$. The partial order $\sqsubseteq$, the binary relation $\sim$, and the operation $\sqcup$ are all extended to $\mathcal{T}^n$ pointwise.

A ternary function, $f: \mathcal{T}^n \rightarrow \mathcal{T}$, is said to be *monotone* when for any $\vec{a} \in \mathcal{T}^n$

and $\vec{b} \in \mathcal{T}^n$ we have

$$\vec{a} \sqsubseteq \vec{b} \Rightarrow f(\vec{a}) \sqsubseteq f(\vec{b})$$

This definition is extended pointwise to vector functions, $\vec{f}: \mathcal{T}^n \rightarrow \mathcal{T}^m$.

The above monotonicity definition is consistent with our use of information content. If a function is monotone, we cannot "gain" any information by reducing the information content of the arguments to the function. In other words, changing some signals from binary values to X will either have no effect on the output values, or it will change some binary values to X.

To express the behavior of a circuit operating over time, we must reason about *sequences* of states. Conceptually, we will consider the state sequences to be infinite, although the properties we will express can always be determined from some bounded length prefix of the sequence. Define a the set $\mathcal{S}^n$ to consist of all sequences $[\vec{a}_0, \vec{a}_1, \ldots]$ where each $a_i \in \mathcal{T}^n$. The relations $\sqsubseteq$ and $\sim$ are extended from vectors to sequences pointwise. That is, two sequences $[\vec{a}_0, \vec{a}_1, \ldots]$ and $[\vec{b}_0, \vec{b}_1, \ldots]$ are ordered (compatible) if and only if each pair $\vec{a}_i$ and $\vec{b}_i$ is ordered (compatible), for all $i \geq 0$.

For vector $\vec{a}$ and sequence S, the expression $\vec{a}S$ denotes the sequence consisting the vector $\vec{a}$ followed by the vectors in S.

3 Circuit Model

The underlying model of a circuit we use is quite simple, as well as general. A circuit $\mathcal{C}$ is a triple $(\mathcal{N}, \vec{Y}, \mathcal{V})$, where $\mathcal{N}$ is a set of nodes (let $s = |\mathcal{N}|$), $\vec{Y}$ is a vector of excitation functions, and $\mathcal{V}$ is a set of symbolic Boolean variables with which parameterized properties of the circuit are to be expressed.

The excitation functions are defined in a non-traditional way. We view them as expressing "constraints" on the values the nodes can take on one time unit later given the current values on the nodes. By constraint we mean specific binary values, whereas the value X indicates that no constraint is imposed. Since the value of an input is controlled by the external environment, the circuit itself does not impose any constraint on the value; hence the excitation of an "input node" is X. More formally, if node n_i corresponds to an input to the circuit then $Y_{n_i}(\vec{a}) = X$ for every $\vec{a} \in \mathcal{T}^s$. Nodes that do not correspond to inputs are called *function nodes.* For a function node n_i the excitation function is a monotone ternary function $Y_{n_i}: \mathcal{T}^s \rightarrow \mathcal{T}$ determined by the circuit topology and functionality.

State sequences are useful when reasoning about circuit behaviors. However, not all state sequences represent possible behaviors of a circuit. The excitation

functions generally restrict the possible state sequences significantly. We formalize this property by introducing the concept of a circuit trajectory. Given a circuit C and an arbitrary sequence $[\vec{a}_0, \vec{a}_1, \ldots] \in \mathcal{S}^s$ we say that the sequence is a *circuit trajectory* if and only if

$$\vec{Y}(\vec{a}_i) \sqsubseteq \vec{a}_{i+1} \text{ for } i \geq 0.$$

The set of all trajectories of circuit C is denoted $S(C)$. The above rule for trajectories is consistent with our definition of an excitation function, i.e., a function computing a constraint on the possible value of a node one time unit later. Thus if the current excitation of a node is binary, say a, then the node must take on the value a in the next state in a valid trajectory. On the other hand, if the excitation is X, then the node value is not constrained.

4 Specification Language

Our specification language describes a property of the circuit as an *assertion* of the form $[A \Longrightarrow C]$, where both A and C are *symbolic trajectory formulas* expressing constraints on the circuit trajectory.

Before we can define our language, we need to introduce some notation and definitions. If $\mathcal{V}$ is a set of symbolic Boolean variables then an *interpretation*, ϕ, is a function $\phi: \mathcal{V} \rightarrow \mathcal{B}$ assigning a binary value to each variable. Let Φ be the set of all possible interpretations, i.e., $\Phi = \{\phi: \mathcal{V} \rightarrow \mathcal{B}\}$. A *domain constraint*, $\mathcal{D} \subseteq \Phi$, defines a restriction on the values assigned to the variables. We will denote such domain constraints by Boolean expressions. That is, let E be a Boolean expression over elements of $\mathcal{V}$.[†] This expression defines a Boolean function $e: \Phi \rightarrow \mathcal{B}$ and thus denotes the domain constraint $\mathcal{D} = \{\phi | e(\phi) = 1\}$. The set of all interpretations Φ is denoted by the Boolean function 1, defined as yielding 1 for all interpretations. Expressing domain constraints by Boolean expressions allows us to compactly specify many different circuit operating conditions with a single formula.

4.1 Symbolic Trajectory Formulas

A trajectory formula expresses a set of constraints on a circuit trajectory. When the formula contains Boolean expressions, each interpretation of the variables yields a different set of constraints. A *step-level* symbolic trajectory formula is defined recursively as:

1. **Constants**: TRUE is a trajectory formula.

[†]For the sake of brevity, we omit a formal syntax of Boolean expressions. Any standard expression syntax suffices.

2. **Atomic propositions:** for $n_i \in \mathcal{N}$ both $(n_i = 1)$ and $(n_i = 0)$ are trajectory formulas.
3. **Conjunction:** $(F_1 \wedge F_2)$ is a trajectory formula if F_1 and F_2 are trajectory formulas.
4. **Domain restriction:** $(E \rightarrow F)$ is a trajectory formula if E is a Boolean expression over $\mathcal{V}$ and F is a trajectory formula.
5. **Next time:** $(\mathbf{X}_s F)$ is a trajectory formula if F is a trajectory formula.

We say that a formula is *instantaneous* when it does not contain any next time operator $\mathbf{X}_s$. For convenience, we often drop parentheses when the intended precedence is clear.

The truth of a formula F is defined relative to a circuit, an interpretation ϕ of the variables in $\mathcal{V}$, and a circuit trajectory. The truth of F, written $C, \phi, S \models F$, is defined recursively. In the following, assume that both S and $\vec{a}S$ are trajectories of C.

1. $C, \phi, S \models \text{TRUE}$ holds trivially.
2. (a) $C, \phi, \vec{a}S \models (n_i = 1)$ iff $a_i = 1$.
   (b) $C, \phi, \vec{a}S \models (n_i = 0)$ iff $a_i = 0$.
3. $C, \phi, S \models (F_1 \wedge F_2)$ iff $C, \phi, S \models F_1$ and $C, \phi, S \models F_2$
4. $C, \phi, S \models (E \rightarrow F)$ iff $e(\phi) = 0$ or $C, \phi, S \models F$, where e is the Boolean function denoted by the Boolean expression E.
5. $C, \phi, \vec{a}S \models \mathbf{X}_s F$ iff $C, \phi, S \models F$.

For an instantaneous formula, its truth can be defined relative to a single state. For instantaneous formula F, the notation $C, \phi, \vec{a} \models F$ indicates that F holds under interpretation ϕ for state $\vec{a}$. A formal definition of this notation can be derived by a straightforward adaptation of rules 1–4 above.

4.2 Assertions

Our verification methodology entails proving *assertions* about the model structure. These assertions are of the form $[A \Longrightarrow C]$, where the *antecedent* A and the *consequent* C are trajectory formulas. The truth of an assertion is defined relative to a circuit C and an interpretation ϕ. Unlike a formula, however, an assertion is considered true only if it holds for all trajectories. That is, $C, \phi \models [A \Longrightarrow C]$,

when for every $S \in \mathcal{S}(C)$ we have that $C, \phi, S \models A$ implies that $C, \phi, S \models C$. Given a circuit and an assertion, the task of our checking algorithm is to compute the Boolean function expressing the set of interpretations under which the assertion is true. For most verification problems, this should simply be the constant function **1**, i.e., the assertion should hold under all variable interpretations.

We have intentionally chosen to introduce only a heavily restricted trajectory formula syntax for our base logic. By imposing these restrictions, we can guarantee the following key property:

Proposition 1 *For any trajectory formula F, and any interpretation ϕ, one of the following cases must hold:*

1. *There is no trajectory $S \in \mathcal{S}(C)$ for which $C, \phi, S \models F$, or*
2. *There exists a unique trajectory $S_{F,\phi} \in \mathcal{S}(C)$ such that for every $S \in \mathcal{S}(C)$ we have $C, \phi, S \models F$ if and only if $S_{F,\phi} \sqsubseteq S$.*

In the first case above, we say that the formula F is not satisfiable under interpretation ϕ. In the second case, we refer to the sequence $S_{F,\phi}$ as the *weakest trajectory* satisfying formula F under interpretation ϕ.

Note that this proposition expresses a very strong property of our logic. It demonstrates the reason why we can verify an assertion by simulating a single symbolic sequence, namely the one encoding the weakest trajectories allowed by the antecedent for every interpretation. It is stronger than the simple monotonicity condition that if $C, \phi, S \models F$ and $S \sqsubseteq S'$, then $C, \phi, S' \models F$.

The logic, as described above, is convenient for deriving the underlying theory. Unfortunately, expressing "interesting" assertions about real circuits using only the constructs above is very tedious. Two shortcomings make using the logic cumbersome: the fine granularity of the timing, and the lack of more powerful logical constructs. It is convenient to add extensions that do not add any expressive power, but make it easier to write assertions.

This basic structure of starting with a minimal basic logic and then adding more elaborate structures as extensions also mirrors our current implementation. The implementation consists of two parts. The underlying logic, with some few extensions, is taken care of by our modified version of the COSMOS symbolic switch-level simulator. The syntactic extensions are supported by a front-end written in SCHEME. The user writes SCHEME code that, when evaluated, generates a file of low-level simulation commands which are then evaluated by the simulator.

5 Symbolic Simulation

In creating a symbolic model, we extend the scalar model defined in terms of the binary and ternary domains $\mathcal{B}$ and $\mathcal{T}$, to one defined in terms of binary- and ternary-valued functions over the variables $\mathcal{V}$. Define the symbolic domain $\mathcal{B}(\mathcal{V})$ (respectively, $\mathcal{T}(\mathcal{V})$) as denoting the set of functions mapping an interpretations in Φ to $\mathcal{B}$ (resp., $\mathcal{T}$). More formally $\mathcal{B}(\mathcal{V}) = \{f: \Phi \rightarrow \mathcal{B}\}$ and $\mathcal{T}(\mathcal{V}) = \{f: \Phi \rightarrow \mathcal{T}\}$. We then extend the operations defined over scalar values to create a symbolic algebra.

We can also extend the vector and sequence algebra defined over scalar values to their counterparts defined over symbolic values. That is, define the vector domain $\mathcal{T}(\mathcal{V})^n$ as

$$\mathcal{T}(\mathcal{V})^n \;=\; \{\langle a_1, \ldots, a_n\rangle | a_i \in \mathcal{T}(\mathcal{V})\}.$$

In implementing a symbolic simulator, we in effect extend the excitation function $\vec{Y}$ to the symbolic domain as $\vec{Y}: \mathcal{T}(\mathcal{V})^s \rightarrow \mathcal{T}(\mathcal{V})^s$. For $\vec{a} \in \mathcal{T}(\mathcal{V})^n$, let $\vec{a}(\phi) \in \mathcal{T}^n$ denote the vector with each element i equal to $a_i(\phi)$. In this way, we can view the symbolic vector $\vec{a} \in \mathcal{T}(\mathcal{V})^n$ either as a vector of symbolic elements, or as a symbolic value which for a given interpretation yields a scalar vector.

We extend most operations from scalar to symbolic domains in a uniform way. Consider an operation $op: \mathcal{D}_1 \times \mathcal{D}_2 \rightarrow \mathcal{D}_3$, defined over vectors, single elements, or a combination of the two. Its symbolic counterpart $op: \mathcal{D}_1(\mathcal{V}) \times \mathcal{D}_2(\mathcal{V}) \rightarrow \mathcal{D}_3(\mathcal{V})$ is defined such that for all $a \in \mathcal{D}_1(\mathcal{V})$ and $b \in \mathcal{D}_2(\mathcal{V})$, we have $(a\ op\ b)(\phi) = a(\phi)\ op\ b(\phi)$. We use this method to extend the ternary algebraic operations $\cdot_t$, $+_t$, and $-^t$, as well as the operation $\sqcup$.

When extending a relation R symbolically, we define the result to be a function specifying the interpretations under which its arguments are related. That is, given a binary relation $R \subseteq \mathcal{D}_1 \times \mathcal{D}_2$, define $R: \mathcal{D}_1(\mathcal{V}) \times \mathcal{D}_2(\mathcal{V}) \rightarrow \mathcal{B}(\mathcal{V})$ as $(a\ R\ b)(\phi) = 1$ if and only $a(\phi)\ R\ b(\phi)$. We use this method to define operations $\sim$ and $\sqsubseteq$ over both single elements and vectors.

We require one operation that is extended to vectors in a nonstandard way. Define the infix operator $?: \mathcal{B} \times \mathcal{T} \rightarrow \mathcal{T}$ as $a\ ?\ b$ equals b if a is 1, and equals X otherwise. When extending this operation to vectors, only the second argument is vector-valued. That is the operation $?: \mathcal{B} \times \mathcal{T}^n \rightarrow \mathcal{T}^n$ is defined as $(a\ ?\ \vec{b})_i = a\ ?\ b_i$. This operation is then extended symbolically in the manner described above.

As a final operation, we define a variant of the join operation that is defined even when for some $\phi \in \Phi$, we have $\vec{a}(\phi) \not\sim \vec{b}(\phi)$. When using this operation, we will separately keep track of the conditions under which the arguments are

compatible. Define the operation $\overset{*}{\sqcup}: \mathcal{T}(\mathcal{V})^n \times \mathcal{T}(\mathcal{V})^n \rightarrow \mathcal{T}(\mathcal{V})^n$ as

$$(\vec{a} \overset{*}{\sqcup} \vec{b})(\phi) = \begin{cases} \vec{a}(\phi) \sqcup \vec{b}(\phi), & \vec{a}(\phi) \sim \vec{b}(\phi) \\ \vec{X}, & \text{otherwise} \end{cases}$$

where $\vec{X}$ denotes a vector with all elements equal to X.

5.1 Translating Instantaneous Formulas to Symbolic Vectors

Given the above definitions and an instantaneous formula F, we derive a "domain" function OK_F, and a "weakest" symbolic vector $\vec{a}_F$ as follows:

1. If F is TRUE then $OK_F = 1$, and $\vec{a}_F = \langle X, \ldots, X \rangle$.
2. (a) If F is $(n_i = 1)$ then $OK_F = 1$, and $\vec{a}_F = \langle X, \ldots, X, 1, X, \ldots, X \rangle$, where the 1 is in position i.
   (b) If F is $(n_i = 0)$ then $OK_F = 1$, and $\vec{a}_F = \langle X, \ldots, X, 0, X, \ldots, X \rangle$, where the 0 is in position i.
3. If F is $(F_1 \wedge F_2)$ then $OK_F = OK_{F_1} \cdot OK_{F_2} \cdot (\vec{a}_{F_1} \sim \vec{a}_{F_2})$, and $\vec{a}_F = \vec{a}_{F_1} \overset{*}{\sqcup} \vec{a}_{F_2}$.
4. If F is $(E \rightarrow F_1)$ then $OK_F = \overline{e} + OK_{F_1}$, and $\vec{a}_F = e \ ? \ \vec{a}_{F_1}$, where e is the Boolean function denoted by the expression E.

The following proposition summarizes the main properties of OK_F and $\vec{a}_F$.

Proposition 2 *Given a circuit C, let F be an instantaneous formula and OK_F and $\vec{a}_F$ be derived as above. Then $OK_F(\phi) = 1$ iff there exists some state $\vec{b} \in \mathcal{T}^s$ such that $C, \phi, \vec{b} \models F$. Furthermore, if $OK_F(\phi) = 1$, then $C, \phi, \vec{b} \models F$ iff $\vec{a}_F(\phi) \sqsubseteq \vec{b}$.*

5.2 Checking Assertions

Our first step in verifying an assertion is to rewrite the antecedent and consequent into a normal form where all next-time operators are collected together. It is easy to show that a trajectory formula F can be rewritten into $F_0 \wedge \mathbf{X}_s F_1 \wedge \mathbf{X}_s^2 F_2 \wedge \ldots \wedge \mathbf{X}_s^{k-1} F_{k-1}$, for some $k \geq 1$, where each F_i is instantaneous. Note that some of the F_i's might be the trivial formula TRUE. Note also that such a sequence can be extended by appending $\mathbf{X}_s^i$TRUE for $i \geq k$. Hence, without any loss of generality, we will henceforth assume that the antecedent and the consequent in an assertion are trajectory formulas in normal form containing the same number of terms.

Given an assertion $[A \implies C]$ of the form

$$[A_0 \wedge \mathbf{X}_s A_1 \wedge \ldots \wedge \mathbf{X}_s^{k-1} A_{k-1} \implies C_0 \wedge \mathbf{X}_s C_1 \wedge \ldots \wedge \mathbf{X}_s^{k-1} C_{k-1}]$$

define a sequence of symbolic ternary vectors $\vec{x}_0, \ldots, \vec{x}_{k-1}$ as follows:

$$\vec{x}_i = \begin{cases} \vec{a}_{A_0}, & i = 0 \\ \vec{Y}(\vec{x}_{i-1}) \overset{*}{\sqcup} \vec{a}_{A_i}, & i > 0. \end{cases}$$

Define the Boolean function $OK_A = \Pi_{0 \leq i < k}\, OK_{A_i}$, where Π denotes Boolean product. This function yields 0 for those interpretations for which the antecedent contains some internal inconsistency. For example, the formula $A = (n_i = a) \wedge (n_i = b)$ would have $OK_A = \overline{a \oplus b}$, because this formula cannot be satisfied when $\phi(a) \neq \phi(b)$. Define the Boolean function $Traj = \Pi_{1 \leq i < k}[\vec{Y}(\vec{x}_{i-1}) \sim \vec{a}_{A_i}]$. This function yields 0 for those interpretations where an incompatibility arises in the trajectory.

We can show that A is satisfiable under some interpretation ϕ if and only if $OK_A(\phi) \cdot Traj(\phi) = 1$. Furthermore, we can extend the sequence $\vec{x}_0, \ldots, \vec{x}_{k-1}$ to be an infinite sequence by defining $\vec{x}_i = \vec{Y}(\vec{x}_{i-1})$ for all $i \geq k$. It can then be shown that for interpretation ϕ the sequence $\vec{x}_0(\phi), \vec{x}_1(\phi), \ldots$ is the weakest trajectory satisfying A under interpretation ϕ. This construction then provides a proof of Proposition 1. This demonstrates how our symbolic simulator can set up the weakest allowable conditions allowed by the antecedent under all possible interpretations.

To check the consequent, define the Boolean function $OK_C = \Pi_{0 \leq i < k}\, OK_{C_i}$. This function yields 0 for those interpretations for which the consequent contains some internal inconsistency. Finally, define the Boolean function $Check = \Pi_{0 \leq i < k}[\vec{a}_{C_i} \sqsubseteq \vec{x}_i]$. This function yields 0 for those interpretations where some trajectory satisfying the antecedent may violate the consequent.

Now define $OK_{[A \implies C]}$ as: $\overline{OK_A} + \overline{Traj} + (OK_C \cdot Check)$. Informally, this equation states that the assertion is true under those interpretations for which the antecedent is unsatisfiable (due either to internal inconsistencies or to an incompatibility in the trajectory), as well as those for which the consequent holds (i.e, it is both internally consistent and is satisfied.)

The main result of this paper is captured in the following theorem:

Theorem 1 *Given a circuit C and an assertion $[A \implies C]$ let $OK_{[A \implies C]} \in \mathcal{B}(\mathcal{V})$ be derived as above. Then*

$$C, \phi \models [A \implies C] \quad \textit{if and only if} \quad OK_{[A \implies C]}(\phi) = 1.$$

Hence, determining whether a circuit satisfies $[A \implies C]$ is reduced to determining whether $OK_{[A \implies C]} = 1$.

6 Conclusions

In terms of mathematical sophistication, the problem solved by our algorithm is far less ambitious than what is attempted by full-fledged temporal logic model checkers. However, we believe that our language is rich enough to be able to describe many important properties of a circuit and to provide a direct path by which such properties may be automatically verified. By keeping the goals of our verifier simple, we obtain an algorithm that is capable of dealing with much larger circuits. We are currently applying these ideas to larger and more complex circuits.

References

[1] D. L. Beatty, R. E. Bryant, and C.-J. H. Seger, "Synchronous Circuit Verification by Symbolic Simulation: An Illustration," *Sixth MIT Conference on Advanced Research in VLSI*, 1990.

[2] S. Bose, and A. L. Fisher, "Automatic Verification of Synchronous Circuits using Symbolic Logic Simulation and Temporal Logic," *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, 1989, pp. 759–764.

[3] R. E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler, "COSMOS: a Compiled Simulator for MOS Circuits," *24th Design Automation Conference*, 1987, 9–16.

[4] R. E. Bryant, "Boolean Analysis of MOS Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-6, No. 4 (July, 1987), 634–649.

[5] R. E. Bryant, "Formal Verification of Memory Circuits by Switch-Level Simulation," To appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1990.

[6] J. A. Brzozowski, and M. Yoeli. "On a Ternary Model of Gate Networks." *IEEE Transactions on Computers C-28*, 3 (March 1979), 178–183.

[7] C-J. Seger, and R. E. Bryant, "Modeling of Circuit Delays in Symbolic Simulation", *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, 1989, pp. 625–639.