Symbolic Simulation—Techniques and Applications[*]

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

Abstract

Symbolic simulation involves evaluating circuit behavior using special symbolic values to encode a range of circuit operating conditions. In one simulation run, a symbolic simulator can compute what would require many runs of a traditional simulator. Symbolic simulation has applications in both logic and timing verification, as well as sequential test generation.

The concept of symbolic simulation has been discussed for over 10 years, but early attempts had only limited success. The recent introduction of more powerful, algorithmic methods of symbolic manipulation have had a major impact on the classes of circuits and properties that can be evaluated symbolically.

1 Introduction

A single run of a conventional simulator provides limited information about the behavior of a digital system. It determines only how the circuit would behave for a single initial state, input sequence, and set of circuit parameters. Characterizing the system for all possible operating conditions by this means is at best impractical and at worst impossible.

Many CAD tasks require more extensive information than can be obtained by a single simulation run. For example, the formal verification of a design requires showing that the circuit will behave properly for all possible initial states and input sequences. Automatic test generation requires selecting a subset from among the set of all input sequences that will detect a given set of faults. Clearly, conventional simulation is of little use for such tasks. Most CAD researchers have solved these problems by means other than simulation. For example, the most common approach to formal verification is by theorem proving [1]. The most common approach to test generation is by combinatorial search, such as the D-algorithm [13].

[*]This research was supported by the Defense Advanced Research Projects Agency, ARPA Order Number 4976, by the Semiconductor Research Corporation under contract 90-DC-068, and by the National Science Foundation under grant number MIP-8913667.

Some tasks that cannot be solved effectively by conventional simulation become tractable by extending the simulator to operate over a *symbolic* domain. Symbolic simulation involves introducing an expanded set of signal values and redefining the basic simulation functions to operate over this expanded set. This enables the simulator to evaluate a range of operating conditions in a single run.

As a simple example, Roth's D-calculus [13] can be viewed as a form of symbolic evaluation, although it is not used for simulation. The D-calculus allows one to simultaneously characterize both faulty and fault-free behavior. It does this by redefining the functions of basic logic gates to operate over an expanded set of values $\{0, 1, D, \overline{D}\}$. This symbolic extension is small—it covers only two different circuit operating conditions—but the basic idea is there.

As a second example, concurrent fault simulation [15] can be viewed as a form of symbolic simulation. This symbolic extension allows one to characterize the behavior of the circuit under fault-free and many different faulty conditions simultaneously. The fault list data structures can be viewed as representing elements of a symbolic domain encoding the value of a signal in the good and all faulty machines. The list manipulations and gate evaluations performed while processing the input lists for a logic gate can be viewed as implementing an extension of the gate function to this symbolic domain. Although this is an unconventional view of concurrent fault simulation, it serves to demonstrate the value of *data abstraction.* That is, we can distinguish between the abstract domain over which the simulation is performed versus the detailed data structures used to represent elements of this domain.

The above two examples illustrate how well-known CAD algorithms can be viewed as making use of symbolic evaluation. In this paper, we will describe the basic principles of symbolic simulation and some novel ways it can be used.

2 Historical Perspective

As already illustrated, the basic principles of symbolic simulation have implicitly been used by CAD programs for many years. The term was introduced explicitly in the late 1970's by researchers at IBM interested in reasoning about circuits represented at the register-transfer level [6, 9]. Their approach drew on techniques developed for reasoning about software by symbolic execution. That is, to evaluate the effect of a sequence of circuit operations, they introduced symbolic values to represent both the possible initial contents of the circuit registers as well as the possible data values applied to the inputs. They extended the primitives of the simulator to operate over expressions involving these symbolic values.

Other researchers continued this work into the early 1980's [8], but activity died off within a few years. People discovered that this form of symbolic evaluation was not powerful enough for reasoning about overall circuit behavior. For example, if the simulator reached a conditional branch while evaluating a sequence of control operations, it would simply evaluate both paths of the branch, but label the outcomes with the conditions under which each branch would be taken. After simulating a number of such branches, the expressions would become too large and cumbersome to use effectively. Looping constructs proved even more intractable, since in general the simulator could not determine the conditions under which the loop would terminate.

The success of this early work was limited by the weakness of the symbolic manipulation methods. During the course of a symbolic simulation, these programs built up algebraic expressions that closely reflected the evaluations that had been performed. Consider, for example how such a program would simulate the effect of adding the contents of two registers and then adding this to the contents of a third register. It would first introduce symbolic values A, B, and C, to indicate the register values. After simulating the two operations, the program would produce an expression $E_1 = (A + B) + C$. Now suppose that another sequence of operations summed these numbers in a different order to produce an expression $E_2 = B + (A + C)$, and that a conditional branch were taken based on a comparison between E_1 and E_2. For such a simple example, a simple algebraic manipulator could exploit the laws of commutativity and associativity to show that the two expressions are always equal. For more complex cases, however, it becomes very difficult to detect such properties in the symbolic expressions.

3 Recent Activity

In the past few years, there has been a renewed interest in symbolic simulation. These recent efforts have been characterized by a more algebraic approach to the problem. That is, rather than build up symbolic expressions based on the evaluation sequence, these programs use representations that are based more strongly on the underlying symbolic domain. The signal values are represented in ways that facilitate the evaluation of the simulation functions over the extended domain. This algebraic approach allows the symbolic simulator to reason more effectively about the actual circuit function.

Several different algebras have been used depending on the class of problems the symbolic simulator is intended to solve.

3.1 Boolean Algebraic Approaches

Boolean algebra is the most natural domain for reasoning about digital circuits, since it directly reflects the binary nature of digital signals. Expressing simulation operations in Boolean algebra also leads directly to a symbolic formulation. That is, if we introduce a set of Boolean variables, then we can let our symbolic domain be the set of Boolean functions over these variables. We then redefine the logic operations AND, OR, and NOT to operate over functions rather than over the values 0 and 1. This gives us an algebra that satisfies all of the laws of a Boolean algebra. Thus, any simulation step that can be expressed in terms of Boolean operations can, in principle, be evaluated symbolically.

As an example, our own work expresses all aspects of a switch-level simulation algorithm in terms of Boolean operations [4]. We use Ordered Binary Decision Diagrams (OBDDs) [3] to represent the Boolean functions created and manipulated by the simulator. This representation has the advantage that it is *canonical*, i.e., each function has a unique representation. Furthermore, the representation is reasonably compact for many of the functions encountered in evaluating digital circuit functions.

Let us return to the example of the register transfer operations described above. Rather than representing the contents of a register with a single integer-valued symbol A, we would represent it as a vector of Boolean-valued symbols $a_{n-1}, \ldots, a_0$, where n is the width of the register. In doing this, we exploit the fact that actual hardware works with finite precision numbers, and hence we do not need to reason about arbitrary integers. This is fortunate, since many properties of the integers are undecidable. As simulation progresses, the circuit signals are expressed in terms of Boolean functions over these symbols. Thus, in summing the contents of the 3 registers, we would derive n OBDDs (one for each bit of the sum) over the $3n$

variables: $a_{n-1}, \ldots, a_0$, $b_{n-1}, \ldots, b_0$, and $c_{n-1}, \ldots, c_0$. While this might seem more cumbersome than constructing the expression $(A+B)+C$, it has the advantage that we would derive the exact same OBDDs regardless of the way these values are summed. Thus, our symbolic manipulator is able to detect all possible relations among the signal values in the circuit.

A Boolean algebraic approach solves many of the problems encountered during early attempts at symbolic simulation. For example, conditional branches cause no problems. We simply follow both branches and combine the two results with a "multiplexor" function. Even the function resulting from a looping construct can be constructed by simply simulating the evaluation of the loop repeatedly until it "terminates," i.e., the conditional expression enabling loop execution evaluates to the constant function 0. In essence, the simulator evaluates the loop for the worst case number of times.

It is obvious that Boolean algebra can be used for reasoning about two-valued digital models. Many digital circuit models, however, are based on multi-valued signal domains. Rather than developing new symbolic representations for these signal domains, we can often encode the elements with multiple Boolean values. Our register transfer example above, for instance, illustrated an encoding of a finite precision integer by a set of Boolean values according to a binary representations. Similarly, our own simulator encodes three-valued signals by pairs of Boolean values. That is, each signal $y \in \{0, 1, X\}$, is represented by a pair of Boolean signals $y.H$ and $y.L$, according to the following encoding:

y	$y.H$	$y.L$
1	1	0
0	0	1
X	1	1

The above encoding is extended symbolically by representing each signal value computed by the simulator as a pair of OBDDs. All of the operations required by the simulator can then be implemented by Boolean manipulation.

As another example of Boolean encoding, researchers at Kyoto University [12] use symbolic simulation to model the behavior of a digital circuit in which each gate can have a range of possible delays. By considering only bounded, integer-valued delays, they can encode each delay as a series of Boolean values using a binary representation. By using Boolean manipulation during the simulation, they can then derive the detailed behavior of the circuit for all possible combinations of gate delays.

Powerful symbolic Boolean manipulation adds an entirely new dimension to the reasoning power of a symbolic simulator. We and others have found it possible to incorporate a number of features into the simulation model, including switch-level models and more detailed timing models [14]. The only requirement is to recast the simulation algorithm into a form that involves only Boolean operations.

3.2 Reasoning about Continuous Systems

For reasoning about digital circuits in which the circuit parameters may vary continuously, it becomes impossible to use Boolean encodings. We require symbolic methods that can deal with infinite domains. Typically, as we move to algebras over infinite domains, the problems quickly become intractable. Nevertheless, symbolic simulators for these tasks can sometimes exploit particular properties of the underlying system to provide useful reasoning capabilities.

For example, prior to their use of Boolean encodings, researchers at Kyoto University considered digital models in which each gate delay can vary continuously between a minimum and maximum value [11]. During the course of a simulation, the signal changes occurring on each line depend on the relative values of the delay sums along different paths in the circuit. Fortunately, these relations can all be expressed as linear constraints, and hence algorithms developed to solve the linear programming problem can be used to determine the conditions under which signal transitions may actually occur. Needless to say, however, this modeling required far more computation than their later version based on integer-valued delays.

Researchers in Belgium have developed a symbolic simulator for modeling analog circuits [10]. Such applications clearly require more powerful symbolic domains than can be represented by Boolean encodings. By linearizing the circuit elements at particular operating points, and by attempting only frequency domain analysis, their program can represent signal values as rational functions in the s (continuous time) or z (discrete time) domain. They can therefore employ algorithms for manipulating rational expressions that have been developed by researchers in the field of symbolic algebra.

As these two experiences show, symbolic simulation can be applied for more complex problem domains where Boolean encodings are impossible. However, the capabilities of such programs are limited in their expressive power and in the size of circuit they can model.

3.3 Weak Signal Algebras

A quite different style of symbolic simulation is to reduce the amount of information that the program attempts to derive. The simulator can then use much simpler and more efficient symbolic algebras but still produce useful results. As an example, researchers at IBM have shown that a conventional 3-valued simulation algorithm can be extended to make a more detailed analysis of the effects of unknown signal values [5]. That is, for each uninitialized state variable i in the simulation, they assign an initial value X_i. Rather than treating each of these initial values as a distinct Boolean variable, however, they view them as unknown, but annotated values. During the course of simulation, they represent each signal as either 0, 1, X_i, $\overline{X_i}$ (the complement of X_i), or X. The value X represents a signal about which nothing is assumed to be known. The simulator can reason to a limited extent about the interactions between unknown values. For example, it can use the identity $X_i \vee \overline{X_i} = 1$. Whenever unknown signals originating from different sources interact, however, an X signal results.

This work indicates how adding a limited amount of symbolic reasoning to a simulator can improve its accuracy in dealing with unknown values without incurring the high overhead of full-fledged symbolic simulation.

4 Applications and Results

The original work on symbolic simulation was directed toward formal circuit verification. This remains the major application for symbolic simulation today. The purpose of formal verification is to prove that the circuit will behave properly for all possible operating conditions. For combinational circuits, verification by symbolic simulation is (conceptually) straightforward. The verifier introduces Boolean variables for each primary input and simulates the circuit operation to compute a Boolean function for each primary output. It then compare these functions with ones derived from the circuit specification.

For sequential circuits, the methodology by which one applies a symbolic simulator to prove circuit correctness is more complex. For cases where the specification and the circuit are sequential circuits using the same state encoding, we can simply verify the combinational logic portion of the circuit. For cases where the specification is given in some other form, or where the state encodings differ, more sophisticated approaches are required. In one recent approach [2], both the circuit and the specification are represented as sequential systems, but with different state encodings. The user must explicitly specify the relation between the two state encodings in terms of an *abstraction function*, mapping a circuit state into a corresponding specification state.

A second application for symbolic simulation has been to automatic test generation. This approach to the test generation problem differs markedly from the more traditional, search-based approaches. In our work [7], we view test generation as proceeding by *symbolic fault simulation*. That is, we simulate the good and faulty circuits over a set of symbolic patterns, effectively evaluating their behaviors over many different actual patterns. We then use Boolean manipulation to derive a set of input sequences that will cause the good and faulty circuits to produce different outputs. This approach has several advantages over search-based methods. Most significantly, it extends quite naturally to both sequential and switch-level circuits. We have demonstrated the ability of our program to generate tests for a number of moderate-sized sequential circuits, but much further work is needed to make this approach truly practical. The memory required to store the OBDDs representing the good and faulty circuit behaviors limits the scale of circuits that we can handle.

5 Conclusions

Symbolic simulation has already produced practical results in formal circuit verification and automatic test generation. The recent advent of algebraic methods employing powerful symbolic manipulation techniques has greatly enhanced the capabilities of these programs. The future prospects appear quite promising, as researchers try new symbolic domains, new manipulation algorithms, and new applications.

References

[1] H. G. Barrow. Proving the Correctness of Digital Hardware Designs. *VLSI Design* Vol. V, No. 7 (July 1984), 64–77.

[2] S. Bose and A. L. Fisher, “Verifying Pipelined Hardware Using Symbolic Logic Simulation,” *International Conference on Computer Design*, October, 1989.

[3] R. E. Bryant. “Graph-Based Algorithms for Boolean Function Manipulation,” *IEEE Transactions on Computers*, Vol. C-35, No. 8 (Aug., 1986), pp. 677–691.

[4] R. E. Bryant, *et al*, “COSMOS: A Compiled Simulator for MOS Circuits,” *24th Design Automation Conference*, 1987, pp. 9–16.

[5] J. L. Carter, *et al*, “Restricted Symbolic Evaluation is Fast and Useful,” *ICCAD-89*, pp. 38–41.

[6] W. C. Carter, W. H. Joyner, Jr., and D. Brand, “Symbolic Simulation for Correct Machine Design,” *16th ACM/IEEE Design Automation Conference*, 1979, pp. 280–286.

[7] K. Cho, and R. E. Bryant, “Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation,” *26th ACM/IEEE Design Automation Conference*, June, 1989, pp. 418–423.

[8] W. E. Cory, “Symbolic Simulation for Functional Verification with ADLIB and SDL,” *18th ACM/IEEE Design Automation Conference*, 1981, pp. 82–89.

[9] J. A. Darringer, “The Application of Program Verification Techniques to Hardware Verification,” *16th ACM/IEEE Design Automation Conference*, 1979, pp. 375–381.

[10] G. G. E. Gielen, H. C. C. Walscharts, and W. M. C. Sansen, “ISAAC: A Symbolic Simulator for Analog Circuits,” *Journal of Solid-State Circuits*, Vol. 24, No. 6 (December, 1989), pp. 1587–1597.

[11] N. Ishiura, M. Takahashi, and S. Yajima, “Time-Symbolic Simulation for Accurate Timing Verification of Ansynchronous Behavior of Logic Circuits,” *26th ACM/IEEE Design Automation Conference*, 1989, pp. 497–502.

[12] N. Ishiura, Y. Deguchi, and S. Yajima, “Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram,” *27th ACM/IEEE Design Automation Conference*, 1990.

[13] J. P. Roth, *Computer Logic, Testing, and Verification*, Computer Science Press, 1980.

[14] C.-J. Seger, and R. E. Bryant, “Modeling of Circuit Delays in Symbolic Simulation,” *IFIP Workshop on Applied Formal Methods for VLSI Design*, November, 1989.

[15] E. Ulrich, and T. Baker, “The Concurrent Simulation of Nearly Identical Digital Networks”, *IEEE Computer* (April, 1974), pp. 39–44.