5

Verifying linear temporal properties of data insensitive controllers using finite instantiations

R. Hojati
University of California, Berkeley
hojati@eecs.berkeley.edu

D. L. Dill
Stanford University
dill@cs.stanford.edu

R. K. Brayton
University of California, Berkeley
brayton@eecs.berkeley.edu

Abstract

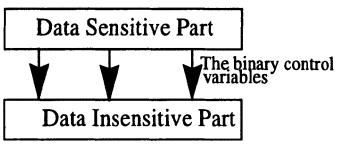
Data insensitive controllers (DICs) are systems where the datapath consists of assignment gates moving the integer data around, and latches storing the data. Memory controllers and communication systems are examples of DICs. In [HB95], it is proved that for DICs the property "when binary variable b becomes true, integer variables x and y are equal" can be proved by down-scaling the integer variables x and y to single-bit binary variables. In this paper, we generalize this notion and consider the problem of verifying properties of DICs in a linear temporal logic whose atomic propositions are finite variables and integer equalities. We show that for this temporal logic, one can always use finite instantiations, although the number of required bits varies with the complexity of the property.

Keywords

Formal verification, Computer-aided design and verification.

1 INTRODUCTION

Data insensitive controllers are an important subset of digital systems in which a controller moves the integer data around. The datapath consists of integer variables, and assignment gates of the form $y := x$ and $z := mux(b, x, y)$, where x, y and z are integer variables, and the binary variable b is driven by the controller. No predicates are applied to the integer variables, and there is no feedback from the datapath into the controller (see figure 1). This concept was first defined in [Wol86], in which it was proved that verifying properties of these systems in a specialized linear temporal logic built up from propositional variables of the form $x = a$, where x is an integer variables and a is a number, can be done using a few data values for the integer variables, i.e. the property holds on the integer system iff it holds on the reduced system. In [HB95], the same concept was formalized in the context of ICS models, and it was proved that for verifying properties of the form "when b becomes true, $x = y$", only two data values suffice. The results of [HB95] were applied to a correctness problem of memory models in [HMLB95].



The data sensitive part generates the control signals for the data insensitive part. Changing the values of variables in the data insensitive part does not affect the values of the data sensitive part.

Figure 1

In this paper, we generalize verifying properties of the form "when b becomes true, $x = y$" and look at a linear temporal logic in which the propositional variables are integer equalities and finite variables of the system (without loss of generality we can assume that all finite variables are binary). We first show that all such properties can be verified by assigning the integer variables a finite domain, whose size is the total number of finite latches and integer inputs of the system. We then consider simple invariance properties of the form $G\phi$, where ϕ is a Boolean combination of propositional variables of the logic (integer equalities and finite variables). For these types of properties, we offer an algorithm which is independent of the system. For example, we show that for verifying the property $G(x = y \vee x = z)$ of any DIC, only two data values are needed, whereas verifying $G(x_1 = x_2 \vee x_3 = x_4 \vee x_5 = x_6)$ requires at most three values.

We then consider another important set of properties, invariance properties with bounded look-ahead. These properties are of the form $G\phi$, where ϕ involves the

temporal next time operator X and Boolean connectives (AND, OR, NOT). We use our algorithm for the invariance case to get bounds for these types of properties as well. The bounds are again independent of the system. For example, we show the property $G(x = y \vee X(x = z))$ can be verified using only two values for integer variables. We also consider liveness properties, and show that without system dependent analysis, one cannot do better than the trivial bound (total number of integer latches and constant creators), since there are systems on which verifying the simple property $F(x = y)$ requires the trivial bound.

Finally, we apply our results to a correctness problem of memory subsystems in microprocessors. In this case, we consider the single, double, and multiple word load and store instructions. In each case, we consider properties that test the correctness of loads. For example, for verifying the correctness of double word loads, we use a property of the form $b \rightarrow ((x = y) \wedge X(x = z))$. Intuitively, this property says that for any arbitrary double word (b signals the beginning of the verification process), the data bus (x) will first contain the first addressed word (y) and then the second addressed word (z). We use our result of section 2.3 to conclude that verifying this property can be done using only two values.

The flow of this paper is as follows. Section 2 contains our main results. In Section 2.1, we show that for our temporal logic, finite instantiations can always be used, and the bound is never worse than the total number of integer latches and constant creators (trivial bound). In Sections 2.2 and 2.3, we give better bounds for invariance properties, without and with bounded look-ahead. In Section 2.4, we present our result for liveness properties. Section 3 details the application of our results to a correctness problem of memory subsystems in microprocessors. Section 4 concludes the paper.

2 MAIN RESULTS

We assume the system is closed, and model integer inputs by ***constant creators*** which produce a new symbolic constant whenever called (a symbolic constant can take on any integer value). We also assume that the initial values for integer latches are arbitrary symbolic constants. Two latches may be assigned the same initial symbolic constant. For simplicity of exposition, we do not define our model in more detail, and refer the interested reader to [HB95].

2.1 General Properties

Definition For a data insensitive controller M, let M_n denote a system where the data insensitive variables of M are replaced by finite variables taking n values (i.e. $\log n$ bits wide).

Lemma 2.1 Let ϕ be a general LTL formula whose propositional variables are integer equalities and finite variables of a DIC M. Let n be the total number of integer latches and constant creators in M. Then, ϕ holds of M iff ϕ holds of M_n.

Proof We use some of the machinery developed in [HB95]. Derive ϕ_b from ϕ by replacing each integer equality with a unique binary variable. Using the Tableau construction ([Wol85]), ϕ_b can be translated into a Buchi automaton B_b with the same language, i.e. $L(\phi_b) = L(B_b)$. Replace the newly introduced binary variables in B_b with their associated integer variables to get the automaton B. We have that ϕ holds of M iff $L(M) \subseteq L(B)$. Construct $\bar{B}$, the complement of B, by replacing the integer equalities with binary variables, complementing the resulting ω-automaton, and substituting back the integer equalities. We have $L(M) \subseteq L(B)$ iff $L(M) \cap L(\bar{B}) = \emptyset$. This check can performed by verifying that the language of the composition of M and $\bar{B}$ is empty. The composition of M and $\bar{B}$ is a system in which the datapath contains assignment operators and equality predicates. In [HB95] it was shown that checking the language emptiness of such a system can be done on a system where integer variables take m values, with m being the number of integer variables. Isles later made the observation that the bound can be reduced to the total number of integer latches and constant creators ([Isl95]). Hence, we have $L(M) \cap L(\bar{B}) = \emptyset$ iff $L(M_n) \cap L(\bar{B}) = \emptyset$ iff $L(M_n) \subseteq L(B)$. We have shown the latter holds iff ϕ holds of M_n. Hence ϕ holds of M iff ϕ holds of M_n (QED).

2.2 Invariance Properties

Let $b_1, b_2, \ldots$ be binary variables, and $x_1, x_2, \ldots$ be integer variables of a given DIC M. Let ϕ be an AND/OR combination of binary variables, their negations, and integer equalities $x_i = x_j$. An example of such properties is $b \rightarrow (x_1 = x_2)$ which is equivalent to $\bar{b} + (x_1 = x_2)$. For now we do not allow general complementation,

although we will shortly generalize to this case as well. Assume we are interested in verifying the property $G\phi$ of M. We would like to do so using finite instantiations with a minimum number of values for integer variables. In this section, we present a heuristic algorithm for choosing the number of data values to use, which is dependent only on the number of integer variables in the property and is independent of the system.

In what follows, let a ***falsification*** for a formula ϕ be an assignment to integer and binary variables in ϕ such that ϕ becomes false. In many of the proofs which follow, we use the concept of the ***source*** for a variable x_i in some state s. This is a constant c_k which has moved around and has ended up in x_i at state s. It was either the initial value of some integer latch, or was created at some previous state by a constant creator. The following lemma establishes a rough upper bound on the number of data values needed.

Lemma 2.2 Let ϕ be an AND/OR formula built from the binary variables, their negations, and integer equalities of a given DIC M. Let n be the number of integer variables in ϕ. Then, $G\phi$ holds of M iff $G\phi$ holds of M_n.

Proof Since $L(M_n) \subseteq L(M)$, if $G\phi$ holds of M, it also holds of M_n. To show the reverse, let $x_1, x_2, \ldots, x_n$ be the integer variables in $G\phi$. Let $\pi = s_1, s_2, \ldots, s_k$ be a path in M such that ϕ does not hold at s_k, which implies $G\phi$ does not hold of π. Let $c_1, c_2, \ldots, c_n$ be the sources for $x_1, x_2, \ldots, x_n$ in s_k. Since ϕ does not hold at s_k, it is possible to assign integer values $i_1, i_2, \ldots, i_n$ to $c_1, c_2, \ldots, c_n$ and leaving the binary variables as they are assigned in s_k such that ϕ is false. If we rename $i_1, i_2, \ldots, i_n$ to take values from $0, 1, \ldots, n-1$, ϕ still remains false. Call this new set $\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_n$. Construct a path $\pi_n = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_k$ in M_n from π such that the binary variables take the same values as in π, $c_1, c_2, \ldots, c_n$ are assigned $\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_n$, and the rest of the constants take arbitrary values. Since the control variables in π and π_n take the same values, we have that $c_1, c_2, \ldots, c_n$ contain $\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_n$ at $\tilde{s}_k$. It follows that $G\phi$ does not hold of π_n (QED).

Lemma 2.2 establishes an upper bound for how many values are needed. We will now improve this bound. For example, we will show that for verifying the formula $G((x = y \vee x = z) \wedge (w = y \vee w = z))$, only two data values suffice. Note that the

bound of lemma 2.2 for this formula is 4 values.

Lemma 2.3 Let ϕ be a disjunction of m integer equalities. Let n be such that $\binom{n}{2} \leq m < \binom{n+1}{2}$. Then, $G\phi$ holds of the DIC M iff $G\phi$ holds of M_n.

Proof It suffices to prove that if $G\phi$ does not hold of M, then $G\phi$ does not hold of M_n. Let $\pi = s_1, ..., s_k$ be a path in M such that ϕ does not hold at s_k. Let $x_1, ..., x_p$ be the variables in ϕ, and $c_1, ..., c_p$ be the sources for $x_1, ..., x_p$ at s_k. Let ϕ_c be the formula which results by replacing each variable by its constant source. Since ϕ is false at s_k, there is no equality of the form $c_i = c_i$ in ϕ_c. Build a graph G with nodes $c_1, ..., c_p$, and an edge for each equality $c_i = c_j$. G contains at most m edges and by lemma 2.4 (below), it can be n-colored. Consider one such n-coloring C. Create an assignment $i_1, i_2, ..., i_p$ to $c_1, ..., c_p$ by choosing values from $\{0, 1, ..., n-1\}$ such that two constants are assigned the same value iff they have the same color in C. This assignment makes ϕ_c false, since the variables in each integer equality are assigned different values. Build a path $\pi_n = t_1, ..., t_k$ in M_n from π by assigning the finite variables the same values as in π, and assigning to $c_1, ..., c_p$ the values $i_1, i_2, ..., i_p$. By construction, ϕ does not hold at t_k,which means $G\phi$ does not hold of M_n (QED)

Example As an example, let ϕ be the formula $(x_1 = x_2) \vee (x_3 = x_4) \vee (x_5 = x_6)$. Let $x_1 = c_1$, $x_2 = c_2$, $x_3 = c_1$, $x_4 = c_3$, $x_5 = c_2$, and $x_6 = c_3$. Substituting for the variables in ϕ the corresponding constants, we get $\phi_c \equiv (c_1 = c_2) \vee (c_1 = c_3) \vee (c_2 = c_3)$. The graph G of lemma 2.3 has 3 edges, and is 3-colorable. Hence, 3 values are needed to verify ϕ. Also, note that 2 values are not sufficient to verify $G\phi$. For example, assume M loads c_1, c_2, c_3 in $x_1, ..., x_6$ as given above, and never changes them again. With only 2 values used for integer variables, $G\phi$ holds of M, whereas with 3 it does not.

Lemma 2.4 Let G be a graph with m edges. Let n be such that $\binom{n}{2} \leq m < \binom{n+1}{2}$. Then, G can be n-colored.

Proof We proceed by induction. If $m = 1$, then the lemma requires that a graph

with one edge be 2-colorable. This is clear as each node can be assigned a different color. Now assume the lemma holds for values up to $m-1$, and prove it for m. If there are no vertices of degree greater than or equal to n (i.e. max degree $< n$), then the lemma follows by ***Vizing's theorem*** ([Viz64]) which states that if the maximum degree of a vertex in a graph is d, then the graph can be colored with $d+1$ colors. Now assume $v \in G$ is a vertex of degree greater n. Assign v a unique color, and remove all edges adjacent to v from G. This new graph has at most $m-n$ edges. Since $\binom{n}{2} \leq m < \binom{n+1}{2}$, we have $\binom{n-1}{2} \leq m-n < \binom{n}{2}$, and by the inductive assumption this new graph is $(n-1)$-colorable, which implies G was n-colorable (QED).

We now present an algorithm which returns the minimum number of data values sufficient to verify the formula $G\phi$, where ϕ is built using AND/OR combinations of integer equalities, binary variables and their negations.

1. *Express ϕ in product-of-sums format, where each conjunct is a disjunction of integer equalities, binary variables and their negations.*
2. *For each conjunct, compute the number of data values needed to falsify the formula using the bound of lemma 2.3.*
3. *Return the maximum among all numbers computed in step 2. This is the number of data values sufficient to verify ϕ.*

The correctness of the algorithm follows by the observation that if ϕ, expressed in POS form, does not hold of M, then one of the conjuncts does not hold of M. The algorithm guarantees there are enough data values to falsify that conjunct.

We now give an algorithm for general Boolean combinations of integer equalities and binary variables by first solving the problem for formulas $G\phi$, where ϕ is a disjunction of integer equality and inequalities (of the form $x \neq y$).

1. *Build equivalence classes for integer inequalities, where if $x_1 \neq x_2$ and $x_2 \neq x_3$ appear in ϕ, then x_1, x_2 and x_3 are all in the same equivalence class. Note that a falsification has to assign the same value to all the variables in an equivalence class.*
2. *From each equivalence class choose a representative variable.*
3. *Delete all integer inequalities from ϕ. Call the resulting formula ϕ_E.*

4. *Replace each variable in ϕ_E by the representative variable from its equivalence class. Call the resulting formula ψ.*
5. *If an equality of the form $x_i = x_i$ is in ψ, ϕ is a tautology, and holds of any system. If not, find the minimum number of data values needed to falsify ψ. This is the number of data values needed to falsify ϕ.*

To get an algorithm for general Boolean combinations of binary variables and integer equalities, express the formula as a product-of-sums, and use the above algorithm to find the minimum number of data values needed for each conjunct (after the binary variables have been deleted from the conjunct). Take the maximum of all these values.

Several remarks are in order. First, note that all bounds on the number of data values needed are based just on the formula, and are independent of the system. Second, Binary Decision Diagrams (BDDs, [Bry86]) can be used to speed up finding a compact POS representation of a Boolean equation.[1] Third, the complexity of our algorithms given a POS representation is polynomial. However, turning a formula into POS representation can be exponential. Also note that any valid POS representation will do, so one that gives the a smaller number may be sought.

2.3 Invariance Properties with Bounded Look-Ahead

In this section, we consider properties of the type $G\phi$, where ϕ is a general Boolean combination of atomic formulas of the form $X^i b$, $X^j x = X^k y$ and $X^j x \neq X^k y$, where b is binary, x and y are integer, X^i denotes i applications of the next time temporal operator X, and i, j and k are non-negative. We call such properties ***invariance properties with bounded look-ahead***, whose correctness at a state s can be established by examining k successors of s, where the parameter k depends only on the formula.

Remark Using the tautologies $X(f \vee g) \equiv Xf \vee Xg$, $X(f \wedge g) \equiv Xf \wedge Xg$, $\overline{Xf} \equiv X\bar{f}$, where f and g are formulas, a more general logic in which the temporal operator X can be applied to sentences can be handled.

The following algorithm returns the number of data values sufficient to verify $G\phi$,

1. To build the POS form for f, build the BDD for f, and create a SOP form for $\bar{f}$ by generating all paths to the leaf 0. By negating this SOP form, we get the POS representation for f.

an invariance property with bounded look-ahead.

1. *For each term of the form $X^i x = X^j y$, introduce two variables x^i and y^j, and replace the integer equality $X^i x = X^j y$ with $x^i = y^j$. Similarly handle inequalities of the form $X^i x \neq X^j y$.*
2. *For each term of the form $X^i b$, where b is a binary variable, introduce a binary variable b^i, and replace $X^i b$ with b^i.*
3. *The resulting formula is an invariance property. Use algorithms of section 2.2 to find the number of data values sufficient to verify this formula.*

Example Consider the formula $G\phi$, where $\phi = b \rightarrow ((x = y) \vee X(z = w))$. The above algorithm produces the formula $G(b \rightarrow ((x = y) \vee (z^1 = w^1)))$, on which the algorithms of the previous section return 2 data values. Hence, 2 data values suffices to prove this property of any DIC. On the other hand if $\phi = (x = y) \vee X(x = y) \vee X(x) = y$, then the algorithm produces the formula $G((x = y) \vee (x^1 = y^1) \vee (x^1 = y))$, on which the algorithm of section 2.2 returns 3 data values.

The following lemma proves the correctness of our algorithm.

Lemma 2.5 Let M be a DIC, and ϕ a general Boolean combination of atomic formulas of the form $X^i b$, $X^i x = X^j y$, and $X^i x \neq X^j y$. Let n be the number retuned by the above algorithm. Then, $G\phi$ holds of M iff $G\phi$ holds of M_n.

Proof Let $b_1, ..., b_k$ and $x_1, ..., x_l$ be the binary and integer variables occurring in ϕ. Let $\tilde{M}$ be obtained from M and ϕ by creating variables of the form b_j^i for each term $X^i b_j$ in ϕ, creating variables x_k^i and x_l^j for each atomic formula $X^i x_k = X^j x_l$, and ensuring in $\tilde{M}$ that $b_j^i = X^i b_j$ and $x_j^i = X^i x_j$. Let $\tilde{\phi}$ be the formula resulting in step 3 of the above algorithm. Then, by construction, ϕ holds of M iff $G\tilde{\phi}$ holds of $\tilde{M}$. By lemma 2.2, $G\tilde{\phi}$ holds of $\tilde{M}$ iff $G\tilde{\phi}$ holds of $\tilde{M}_n$. By construction of $\tilde{M}$ and $\tilde{\phi}$, $G\tilde{\phi}$ holds of $\tilde{M}_n$ iff $G\phi$ holds of M_n. So, we have proved $G\phi$ holds of M iff $G\phi$ holds of M_n as was required (QED).

2.4 Liveness Properties

In this section, we show that if the temporal operator F is used in the formula, then no bound other than the trivial bound can be proved. Recall that the trivial bound is the total number of latches and constant creators in the system.

Lemma 2.6 There is a system M, with the total number of integer latches and constant creators being n, on which the property $F(x = y)$ holds of M_i for $i < n$, but does not hold of M_n.

Proof M has no constant creators, but it has n integer latches $x_1, ..., x_n$, all of which take a different initial symbolic constant, i.e. $c_1, ..., c_n$. Let x and y be two integer outputs of M. At each time step, the system loads into x and y a new pair of values $\{c_j, c_k\}$. For $i < n$, there are two latches which take the same initial value in M_i. Therefore, at some point $x = y$, i.e. $F(x = y)$ holds of M_i. On the other hand, for M_n, $F(x = y)$ does not hold for the trace where distinct initial values are assigned to the integer latches (QED).

3 APPLICATIONS

In this section, we describe an application of the results of the previous sections to a correctness problem of memory subsystems in microprocessors. Many instruction set architectures allow for loading and storing of single words, double words, or multiple words (for an example, see [PowerPC94]). In this section, we show how some correctness properties of these instructions may be verified using results from the previous sections.

Figure 2 shows a typical configuration for a memory subsystem. The environment, which is an abstraction of the fetch-dispatch-execute core of the processor, sends load and store instructions identified by their tags, addresses and data to the memory subsystem. The memory subsystem (after some time) services these instructions. When an instruction is serviced, the memory subsystem puts the instruction's tag on the instruction tag bus. When servicing a load, the data is also placed on the data bus. We assume that the data bus is 32 bits wide. So, double and multiple word loads take several cycles to complete.

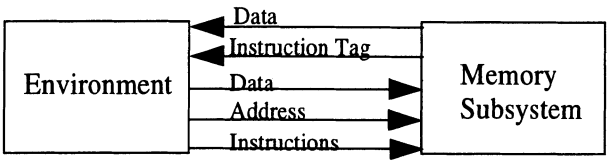


Figure 2

To verify the correctness of the memory subsystem, we assume that the number of memory locations is finite, and augment the environment with a copy of every memory location. This copy keeps track of what the values in memory should be by monitoring the instructions issued by the environment. One simple property to verify is that after any set of instructions, a load returns the correct value. This can be done by having a non-deterministic signal which starts the checking process after an arbitrary load. At this point, the copy of that memory location stops tracking the new values. When the load is serviced, it is checked that the returned value is equal to the copy. This property is of the form $b \rightarrow (x = y)$. This property can be checked using just two values as shown in [HB95].

Similarly, verifying a double word load can be done using a property of the form $b \rightarrow ((x = y) \wedge X(x = z))$, where x represents the data bus, y and z represent the expected first and second words respectively. Noting that when expressed as a product-of-sums, each conjunct has at most one integer equality, and using results of section 2.3, this property is independent of the system being verified, and can be verified using 2 data values.

Some architectures allow for loads with byte reversed ordering. For verifying such instructions, one might use a property of the form $b \rightarrow (((x = y) \wedge X(x = z)) \vee ((x = z) \wedge X(x = y)))$, where x is the data bus, y and z are the expected first and second words. Noting that when expressed as a product-of-sums, each conjunct has at most two integer equalities, and using results of section 2.3, we conclude this property can be verified using only 2 data values.

As a sanity check when doing a multiple word check, one might check that the last word of the load appears on the data bus. This property can be expressed as $b \rightarrow F(x = y)$, where x is the data bus and y is the expected last word of the load. Using results of section 2.4, verifying this property using finite instantiations depends on the system being verified. The upper bound on the sufficient number of data values is the total number of integer latches and constant creators (integer inputs) of the system. However, if one knows how many cycles it takes to get the last word, then F can be replaced by X^i, and better bounds can be obtained.

4 CONCLUSIONS

In this paper, we considered the problem of verifying general linear temporal properties of data-insensitive controllers, where the propositional variables are the finite variables of the controller and integer equalities of the form $x = y$ (using complementation one can get integer inequalities as well). We first showed that all such properties can be verified using finite instantiations, where the size of the domain for integers is at most the total number of finite latches and constant creators. We then considered invariance and invariance properties with bounded look-ahead, and proved bounds which are independent of the system being verified. We then showed that for liveness properties one cannot hope to do much better than the trivial bound (the total number of integer latches and constant creators), since there are systems on which verifying the simple property $F(x = y)$ requires the trivial bound. We finally showed how our results can be applied to a correctness problem of memory subsystems in microprocessors.

5 REFERENCES

[Bry86] R. E. Bryant, "*Graph Based Algorithms for Boolean Function Manipulation*", IEEE Trans. on Computers, C-35(8):677-691, August 1986.

[HB95] R. Hojati, R. K. Brayton, "*Automatic Datapath Abstraction In Hardware Systems*", Conference on Computer-Aided Verification, June 1995.

[HMLB95] R. Hojati, R. Mueller-Thuns, P. Loewenstein, R. K. Brayton, "*Automatic Verification of Memory System Using Language Containment and Abstraction*", Conference on Hardware Description Languages and Their Applications, 1995.

[Isl95] A. Isles, personal communication, 1995.

[PowerPC94] IBM Microelectronics and Motorola, "*PowerPC Microprocessor Family: The Programming Environment*", 1994.

[Viz64] V. G. Vizing, "*On an Estimate of the Chromatic Class of a p-Graph*", (in Russian), Diskret. Analiz., Vol. 3, 25-30 (1964).

[Wol86] P. Wolper, "*Expressing Interesting Properties of Programs*", 13th Annual ACM Symp. on Principles of Prog. Languages, 1986.

[Wol85] P. Wolper, "*The Tableau Method for Temporal Logic: An Overview*", Logique at Anal. 28, pp. 119-136, 1985.

6 BIOGRAPHY

Ramin Hojati obtained his B.S. from Massachusetts Institute of Technology in 1988. From 1988 to 1990, he worked on layout and synthesis tools at Cadence Design Systems. He obtained his M.S. and Ph.D. in computer science in 1992 and 1996, respectively from the University of California, Berkeley. Dr. Hojati has written many research papers in computer-aided design and formal verification, and has been involved in several large scale software developments. Currently, he is a post-doctoral research staff member at UC Berkeley where continues research and advises students. In addition, he recently founded "Ramin Hojati Consulting", which provides consulting in computer-aided design. His research interests include formal verification, logic synthesis and layout of digital systems.

David L. Dill is Associate Professor of Computer Science and, by courtesy, Electrical Engineering at Stanford University. He has been on the faculty at Stanford since 1987. He has an S.B. in Electrical Engineering and Computer Science from Massachusetts Institute of Technology (1979), and an M.S and Ph.D. from Carnegie-Mellon University (1982 and 1987). His primary research interests relate to the theory and application of formal verification techniques to system designs, including hardware, protocols, and software.

Robert Brayton received the BSEE degree from Iowa State University in 1956 and the Ph.D. degree in mathematics from MIT in 1961. From 1961 to 1987 he was a member of the Mathematical Sciences Department of the IBM T. J. Watson Research Center. In 1987 he joined the EECS Department at Berkeley, where he is a Professor and director of the SRC Center of Excellence for Design Sciences. He has authored over 200 technical papers, and six books, "Computer Aided Design: Sensitivity and Optimization", "Logic Minimization Algorithms for VLSI Synthesis", "Integrating Functional and Temporal Domains in Logic Design", "Timed Boolean Functions: A Unified Formalism for Exact Timing Analysis", "Logic Synthesis for Field-Programmable Gate Arrays", "Synthesis of Finite State Machines: Functional Optimization". Dr. Brayton is a member of the National Academy of Engineering, and a Fellow of the IEEE and the AAAS. He received the 1991 IEEE CAS technical achievement award, the 1971 Guilleman-Cauer award, and the 1987 Darlington award. He was the editor of the Journal on Formal Methods in Systems Design from 1992-1996. Past contributions have been in analysis of nonlinear networks, and electrical simulation and optimization of circuits. Current research involves combinational and sequential logic synthesis for area/performance/testability, asynchronous synthesis, and formal design verification.

Acknowledgment

The authors would like to thank the committee members for their insightful comments. We would also like to thank Ken McMillan for useful discussions, and Adrian Isles for his careful reading of a draft of this paper. During this work, the first author was supported by SRC grant 96-DC-324.