AUTOMATIC STATE REDUCTION TECHNIQUES FOR HARDWARE SYSTEMS MODELED USING UNINTERPRETED FUNCTIONS AND INFINITE MEMORY

by

Ramin Hojati, Adrian J. Isles, and Robert K. Brayton

Memorandum No. UCB/ERL M97/53

1 May 1997

AUTOMATIC STATE REDUCTION TECHNIQUES FOR HARDWARE SYSTEMS MODELED USING UNINTERPRETED FUNCTIONS AND INFINITE MEMORY

by

Ramin Hojati, Adrian J. Isles, and Robert K. Brayton

Memorandum No. UCB/ERL M97/53

1 May 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Automatic State Reduction Techniques for Hardware Systems Modeled Using Uninterpreted Functions and Infinite Memory

Ramin Hojati (hojati@eecs.berkeley.edu)
Adrian J. Isles (aji@eecs.berkeley.edu)
Robert K. Brayton (brayton@eecs.berkeley.edu)

Department of Electrical Engineering and Computer Sciences
The University of California, Berkeley, CA 94720-1770

Abstract

The Integer Combinational Sequential (ICS) concurrency model is designed to represent hardware systems at a high level of abstraction using infinite memory, uninterpreted and interpreted integer functions and predicates. According to their semantics, which are natural when dealing with these more abstract entities, many systems which intuitively should have finite reachable states sets, have infinite state spaces. In this paper, we first define an equivalence notion between two states in an ICS model, and then present (and prove correct) a set of operations which identify equivalent states. These optimizations can be used to significantly speed-up reachability computations when dealing with uninterpreted functions and infinite memory. In addition, we present an automatic verification procedure for a set of circuits involving infinite memory, integer address and data, whose addresses and data are independent, by proving the state-reduced reachability computation generates a finite state space. Finally, we give an example for which the state-reduced ICS reachability generates a linear number of states whereas the smallest finite instantiation containing all behaviors has an exponential number of states. The state reductions presented in this paper are implemented in our second generation ICS reachability tool and some experimental results are given.

1 Introduction

Reduced Ordered Binary Decision Diagrams (BDDs) [Bry86] have proven to be very successful in verifying control dominated circuits. BDD based techniques, however, can at best handle a few hundred latches and generally perform poorly when applied to circuits that contain datapaths and memory. In an attempt to overcome the state space explosion due to datapaths, the ***Integer Combinational Sequential*** (ICS) concurrency model was introduced in [HB95]. ICS models allow finite gates and latches to represent control, but abstract the datapath by representing datapath variables as integers, datapath operations as integer operations (using both interpreted and uninterpreted integer functions[1]), and memory operations as abstract operations on infinite memories. Using uninterpreted functions to represent datapath operations such as addition and multiplication can reduce the complexity of a design and its associated verification problems considerably. Using such an abstract representation to model hardware systems is certainly not new. What makes ICS unique, however, is that it allows for automatic state exploration to be performed on systems described in this fashion. Traditional state exploration based verification techniques, such as model checking and language containment, can then be used to verify control

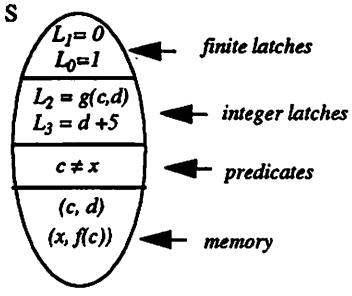
1. All that is known about an operation modeled using an uninterpreted function is that it is, in fact, a function (i.e. if $a = b$ then $f(a) = f(b)$ for an uninterpreted function f). This is contrast to non-deterministic gates, which may have different outputs for the same input assignments.

properties of ICS state graphs. Since the model allows integers and infinite memory, ICS models may have, in some cases, an infinite number of reachable states. In such cases, state exploration can be performed for a finite number of steps (let's say n). Partial verification can then be used to show that there are no error traces up to length n.

There have been several other techniques in the literature that allow for automatic verification of systems modeled at a level of abstraction similar to ICS. In particular, a special form of symbolic simulation, presented by [BD94], has been developed that is fast and allows for the automatic verification of a pipelined microprocessor against its instruction set architecture. Their technique, however, requires the user to specify a superset of the set of reachable states of the system. Finding such ***state-invariants*** can sometimes be extremely difficult for the user [SDB96]. One of the primary advantages of verifying designs in the ICS paradigm is that state invariants are not needed since the reachable set is computed automatically. In addition, ICS models allow for a rich set of properties to be verified, since both language containment and model checking can be used. Another technique that is similar in flavor to ICS is a data structure called Multiway Decision Graphs (MDGs) [Cor93]. An MDG is a decision diagram that is similar to BDDs except that it allows for uninterpreted function symbols to be represented in the decision diagram. MDGs are the only other technique in the literature that the authors are aware of that can represent the set of reachable states and perform fixed point state computations on systems modeled with integer variables and uninterpreted functions. Unlike ICS and [BD94], however, MDGs, cannot represent infinite memory, which is important when trying to verify hardware systems such as microprocessors.

[IHB96] implemented a routine to generate ICS state graphs and presented experimental results on verifying systems modeled using ICS. The results were poor, however, when compared to [BD94], due to an explosion in the number of states of even simple microprocessors. This paper presents new state reductions techniques that can significantly improve the performance of the ICS reachability routine. Below we give an overview of these techniques and some their applications.

Informally, an ICS state $S \doteq (latches, memory, predicates)$ is a set of assignments to finite latches, integer latches, predicates and memory. Assignments to finite latches take from values in their domains. Assignments to integer latches are symbolic expressions called ICS terms (to be formally defined later). Memory is a table of address/value pairs which are also represented using ICS terms. Although, ICS memory is infinite, it is only necessary to explicitly store locations that have been previously written. Predicates define a set of constraints on the ICS terms in S. Figure 1 shows a diagram of an ICS state.



An ICS state consists of a set of assignments to latches and memory, as well as a set of predicates that are valid in the state. Although memory in infinite, only locations that have been previously written to have to be explicitly stored in memory.

Figure 1

One can then define the state exploration of ICS models in a natural way with the following caveats.

1. A state contains the set of assumptions (predicates) made in reaching that state. For example, if constants a and b are compared, then both paths, one where $a = b$ and the other where $a \neq b$ have to be considered. These assumptions must be remembered and become part of the state, since, for example, along the path where we have assumed $a = b$, $f(a) = f(b)$ also holds.
2. When reading from or writing to memory, all possibilities between the accessed address and the addresses currently in memory have to be considered. For example, if there is a read from address a, and the addresses in memory are a_1 and a_2, then three possibilities $a = a_1$, $a = a_2$, and a being distinct from a_1 and a_2 have to be considered.
3. Integer inputs are represented by ***constant creators***. A constant creator returns a new constant whenever called.

Many characteristics of the semantics of ICS models decrease the chance for state sharing. For example, along a path a constant creator has to always return a new constant, and cannot re-use constants; or "extra" predicates and memory

locations which no longer give useful information cannot be deleted. Therefore, many systems which intuitively should have a finite state space have an infinite state space. For example, a system composed of a single control state, which at every cycle produces a new constant has an infinite state space. Intuitively, this system should have only one state.

In this paper, we first introduce a notion of equivalence between two states in an ICS model, and then define a set of optimizations which delete extra information, and hence increase the chance of state sharing. More specifically, two states s_1 and s_2 are ***trace equivalent*** iff the set of traces starting from s_1 and s_2 projected to the set of finite (non-integer) variables are the same. One can then show that the language from s_1 is empty iff the language from s_2 is empty. Since language emptiness is the vehicle for verification in our environment, trace equivalence preserves verification. An example of the importance of automatic state reduction in verifying ICS Models is shown in Figure 2. Using this notion of equivalence, we then show the following operations result in trace equivalent states.

1. Replacing two ***isomorphic*** states, i.e. two states which can be mapped to one another by a renaming of the constants, by one of them.

2. Deleting an equality predicate $a = b$, and replacing all occurrences of the constant b by the constant a (***propagating equalities***).

3. Replacing a term of the form $f(\alpha_1, ..., a, ..., \alpha_n)$, where f is an uninterpreted function, by a new constant if a is not stored in any of the latches, memories, and does not appear in any of the predicates as a term (***simplifying functional terms***).

4. Deleting an equality, inequality, or uninterpreted predicate, which contains a constant not appearing anywhere else in the system (***deleting dangling predicates***).

5. Deleting a memory location $(address, data)$ of the form (a, γ), where a does not appear anywhere else in the system (***deleting dangling memory locations***).

These results are then used to show the following.

1. The language emptiness problem is decidable for ***decoupled data independent memory systems***, in which data and address paths are separate (i.e. addresses cannot be stored as data in memory), and there are no uninterpreted functions or predicates, but there could be an infinite memory.

2. There are cases for which optimized ICS reachability produces a linear number of states, but the smallest finite instantiation has an exponential number of states. For these cases, even after using BDDs or symmetry reduction on the finite instantiation, the running time of verification is still at best quadratic. Hence, at least in an asymptotic sense, ICS models (with their potentially infinite state spaces) can have a computational advantage over finite instantiations and the best known finite state algorithms. Experimental results against the BDD-based tool, VIS [VIS96], confirms this, and is presented here. In practice, we expect that our optimizations, possibly extended by other ones, can have a major impact in reducing the computational complexity of ICS reachability.

It also appears that our optimizations may pave the way to find decision procedures for some open problems of ICS models. Two such open problems follow, where it seems if the problems are decidable, then the optimizations proposed in this paper can be used in conjunction with some new ones to get decision procedures.

1. In [HIKB96], it was shown that if the datapath of an ICS model contains uninterpreted functions and integer equalities, then the reachability problem is undecidable. However, the decidability of the property "when b becomes 1, $x = y$), where b is binary and x and y are integer, of ICS models with only uninterpreted functions in the datapath was left open. Note that for these ICS models, the equality predicate appears only in the property, and not in the system.

2. The decidability of the reachability problem for data independent memory systems is open, where there is an infinite memory, and the datapath contains data movement elements and the equality predicate.

The flow of the paper is as follows. Section 2 describes ICS models and their semantics. Section 3 describes the main technical contributions of the paper. The applications of the results of section 3 are presented in section 4. Some experimental results are presented in section 5. Section 6 concludes the paper, and presents some future directions. Due to lack of space, the proofs of many of the lemmas and theorems, which are rather detailed are presented in the appendix.

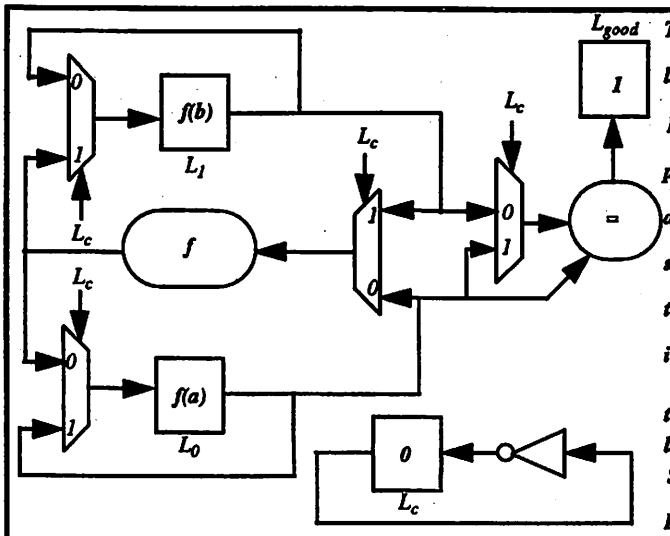
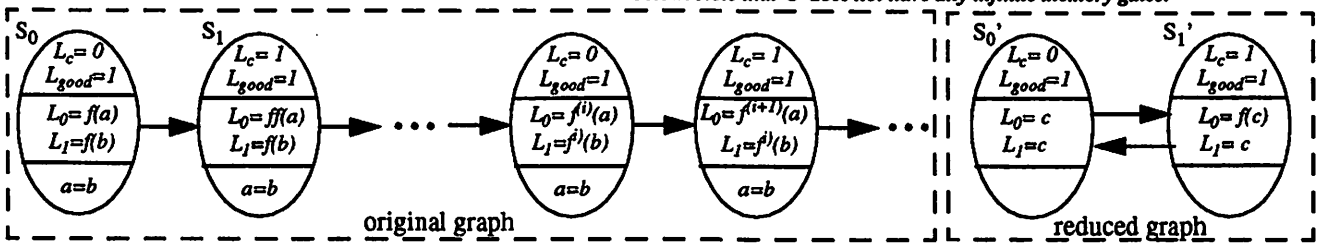


Figure 2

The circuit C to the left has two integer latches L_0, L_1 and two finite latches L_c, L_{good}. Assume that C is in a state S_0 where $L_0 = f(a)$, $L_1 = f(b)$, $L_c = 0$, $L_{good} = 1$ and $P = \{a = b\}$ (P are the predicate assumptions made at S_0). When $L_c = 0$ ($L_c = 1$), C takes a term stored in L_0 (L_1), applies the uninterpreted function f to it and stores results. In addition, when $L_c = 0$, C also checks to see if the terms in L_0 and L_1 are equal and the result of the comparison is stored in L_{good}. C has an infinite set of states, since there are an infinite set of terms $f(a), f(b), \ldots, f^{(i)}(a), f^{(i)}(b), \ldots$ that can be assigned to the latches. Ordinarily, to verify the property that at all states in the future of S_0, $L_{good} = 1$, would require visiting an infinite set of states. However, S_0 is trace equivalent (w.r.t. to control behavior) to any other state S' where $L_0 = f^{(i)}(c)$, $L_1 = f^{(i)}(c)$, $L_c = 0$, $L_{good} = 1$ for some constant c. Thus, the infinite state graph can be automatically reduced to a finite graph. The original and reduced graphs are shown below. Note that C does not have any infinite memory gates.



2 Integer Combinational/Sequential (ICS) Concurrency Model

In this section, both the syntax and semantics of ICS models are reviewed. The reader is encouraged to read [HB95] for a more detailed presentation. ICS is designed to represent systems composed of control, datapath, and memory. Its syntax and semantics is similar to traditional models for representing hardware systems, with the addition of some machinery to reason about non-deterministic gates, integers functions and predicates, and infinite memory.

2.1 Syntax

The primitives are: variables, tables, interpreted functions and predicates, uninterpreted functions and predicates, constant creators, latches, and memory functions.

Variables. Variables are of two types: ***finite*** and ***integer***. Finite variables take values from some finite domain; integer variables take integer values (0, 1, 2, ...).

Tables. A table is a relation defined over a set of finite variables, divided into inputs and outputs. A table is a ***function*** if for every possible input tuple there is at most one output tuple (incompletely specified functions are allowed). Otherwise it is a ***relation***. If a table has only one binary output, and is a function, then it is a ***predicate***.

Interpreted Functions and Predicates. A predefined set of functions and relations over integers is built in. The interpreted functions are: $y := x$, $y := if(b, x)$, $z := mux(b, x, y)$, $z := x + y$, $y := x + c$, where x, y, and z are integer variables, b a binary variable, and c a non-negative integer (0, 1, 2, ...). The interpreted predicates are $y = x$ (equality), $y < x$, $x = c$, $(x \bmod m) = r$, and $(x \bmod m) < r$, where c, r, and m are given non-negative integers.

Uninterpreted Functions and Predicates. These are a set of function and predicate symbols where only their arities and domain variables are given. For example, $f(x_1, x_2)$ may be the specification of an uninterpreted integer function defined

over binary variable x_1 and integer variable x_2. Predicates of the form $x = term$, where x is an integer variable, and *term* is an ICS term are also allowed. An ***ICS term*** is built recursively from numerals, constants, interpreted and uninterpreted functions. Therefore, numerals and constants are ICS terms, and if f is an n-ary function and $t_1, \ldots, t_n$ are ICS terms, then $f(t_1, \ldots, t_n)$ is also an ICS term. Note that ICS terms do not involve any variables or predicates. Examples of ICS terms are c_0, $f(c_0, c_1)$, $g(15, f(c_2, c_3))$, and $c_1 + c_2$. Uninterpreted functions are very useful for abstracting away details of a design that are not needed for verification. This is particularly important when verifying, for example, the control logic of a superscalar microprocessors. Here, verification is independent of the functional units in the datapath, since one only needs to check whether the processor executes each instruction in the proper sequence and writes back the result to the correct location. Therefore, the detailed functionality of the datapath is not relevant and can be replaced with uninterpreted functions.

Constant Creators. A constant creator is a special element with no input, and an integer output. Intuitively, it is a higher-order function, which creates a new constant (i.e. a function with no argument) each time called. A constant creator models an unconstrained integer input.

Latches. A latch is defined on two variables over the same domain: input (or ***next state***) and output (or ***present state***). Present and next state variables may overlap, e.g. when an output of a latch is an input to another. Every latch has a set of initial values, which are a subset of the domain of its variables. If the latch is integer-valued, then the initial value set can either be a finite set of numerals, or a given constant. Predicates can be used in combination with constants to create an infinite set of initial values. For example, to declare that the initial set of a latch is all integers greater than 5, we let the initial value be some constant c_0. In the first state, the output of the latch is input to a predicate $x > 5$, and the machine continues only if the predicate holds. Hence, only those behaviors are allowed where the initial value of the latch is an integer greater than 5.

Memory Functions. Two functions ***read*** and ***write*** are provided with their usual interpretation; *read* is a binary function of a memory and a location; *write* is a ternary function, whose arguments are a memory, a location, and a value. Location and value are variables in the model. Reading a location which has not been written, returns a new constant (like a constant creator).

Definition A ***generalized gate*** is a table, an interpreted or uninterpreted function or predicate, or a constant creator.

Definition ***Data movement operations*** are $x := y$, $z := mux(b, x, y)$, and $y := if(b, x)$, where x, y, z are integer variables, and b is binary.

Every model has only a finite number of variables, latches, and generalized gates. Every variable is the output of exactly one generalized gate or latch. Hence, every input to a generalized gate or latch is the output of some other generalized gate or latch; ICS models are closed. A variable can be input to many generalized gates or latches.

Definition A ***state*** is a triple $(latch, memories, predicates)$, where,

a. *latch* is an assignment of values to the latches. For finite valued latches, the value comes from the domain. For integer valued latches, the value is an ICS term.

b. *memories* is a set of memory elements, where a ***memory*** is a set of pairs of ICS terms, where the first denotes a location and the second a value.

c. *predicates* is a set of ***atomic formulas***, where an atomic formula is any interpreted or uninterpreted predicate applied to ICS terms. Examples are $c_0 < f(c_1)$ and $P(f(c_0, c_1), g(c_0))$. Note that $(c_0 < f(c_1)) \wedge (P(f(c_0, c_1), g(c_0)))$ is not an atomic formula. Intuitively, *predicates* at a state s, is the set of assumptions made to reach s.

Definition Given two ICS terms t_1 and t_2, and two sets of atomic formulas $P = \{P_1, \ldots, P_n\}$ and $Q = \{Q_1, \ldots, Q_m\}$, t_1 ***is equal to*** t_2 ***subject to*** P ***and*** Q, denoted as $t_1|_P = t_2|_Q$ iff the formula

$(P_1 \wedge \ldots \wedge P_n \wedge Q_1 \wedge \ldots \wedge Q_m) \Rightarrow (t_1 = t_2)$ is valid. For example, if $t_1 = x$, $P = \{x > 7, x \leq 8\}$, $t_2 = 8$, and $Q = \emptyset$, then $t_1 = t_2$ subject to P and Q since $(x > 7 \wedge x \leq 8) \Rightarrow (x = 8)$ is valid. The equality of two ICS terms can be decided using the algorithms of [Sho79].

Notation If $P = \{P_1, \ldots, P_n\}$ is a set of predicates, we will use P to denote $P_1 \wedge \ldots \wedge P_n$. For example, $P \rightarrow (b = 1)$ is the formula $(P_1 \wedge \ldots \wedge P_n) \rightarrow (b = 1)$.

Definition Let states $s_1 = (L_1, (M_1^1, \ldots, M_n^1), P_1)$ and $s_2 = (L_2, (M_1^2, \ldots, M_n^2), P_2)$ be given. $s_1 = s_2$ if the following hold.

a. Let l_1^i and l_2^i denote the values of the i-th latch in L_1 and L_2, respectively. If the i-th latch is finite, then $l_1^i = l_2^i$; otherwise, $l_1^i\big|_{P_1} = l_2^i\big|_{P_2}$ must hold.

b. Let $M_k^1[i] = (a_k^1[i], v_k^1[i])$ denote the i-th address/value pair in M_k^1, the k-th memory element of s_1. Then, for each $M_k^1[i]$ there exists $M_k^2[j]$ such that $a_k^1[i]\big|_{P_1} = a_k^2[j]\big|_{P_2}$ and $v_k^1[i]\big|_{P_1} = v_k^2[j]\big|_{P_2}$. Similarly, for each $M_k^2[j]$ there must exist $M_k^1[i]$ such that $a_k^1[i]\big|_{P_1} = a_k^2[j]\big|_{P_2}$ and $v_k^1[i]\big|_{P_1} = v_k^2[j]\big|_{P_2}$.

c. $P_1 \equiv P_2$, i.e. $P_1 \Rightarrow P_2$ and $P_2 \Rightarrow P_1$.

Definition An ***initial state*** is a state $(latch_{init}, \emptyset, \emptyset)$, where $latch_{init}$ is an assignment of an initial value to each latch.

2.2 Operational Semantics of ICS

The operational semantics describes how transitions are made between states of a design modeled using ICS.

Definition A ***gate graph*** H is a directed graph where each node is a generalized gate. $(x, y) \in H$ if some output variable of the generalized gate x is an input to the generalized gate y. A cyclic gate graph is said to contain a ***combinational loop (or cycle)***.

Remark For an acyclic gate graph H, a root node either has no inputs or is a latch.

Given s is a state of the model, and a counter n, representing the number of constants created so far during the symbolic execution of the state space. A state transition is defined by the following algorithm which, given a state $u = (L, M, P)$, assigns a new value to all next state variables (creating L'), and creates a new memory M', a new set of predicates P', and a new counter n'. In this case, we say there is a state transition (u, v), where $v = (L', M', P')$. State transitions are computed as follows.

0. Let $P' = P$, $n' = n$.

1. Let a user-given total order on all memory operations be given. Choose a topological sort O of the gate graph consistent with this memory order.

2. Assign the values given by L to the outputs of the latches.

3. Assign values to the outputs of each generalized gate consistent with its inputs, processing the generalized gates in the topological order O. More precisely, let a generalized gate $g(i, o)$ be given, where i represents the inputs to the gate and o its output.

   a. If g is a table representing the relation $R(i, o)$, then $(i, o) \in R$.

   b. If g is an integer function, then $o = g(i)$, where o and i are ICS terms.

   c. If g is an integer predicate, if $P' \rightarrow (g = 0)$ is valid, let $g = 0$; if $P' \rightarrow (g = 1)$ is valid, let $g = 1$; otherwise let $o = 0$ or $o = 1$, and $P' = P' \cup \{g = o\}$.

   d. If g is a constant creator, then $o = c_{n'}$, where $c_{n'}$ is a fresh constant. Let $n' = n' + 1$.

e. Assume g is a memory read operation with address α, and memory contains a set of address/value pairs (α_i, d_i). If there exists an i such that $P' \rightarrow (\alpha = \alpha_i)$ is valid, then let $o = d_i$ (i.e. read the data from location α_i). If no such α_i exists, then perform one of the following 1) choose an i, where $P' \rightarrow (\alpha \neq \alpha_i)$ is not valid, let $P' = P' \cup \{\alpha = \alpha_i\}$, and $o = d_i$, or 2) for all i where $P' \rightarrow (\alpha \neq \alpha_i)$ is not valid, let $P' = P' \cup \{\alpha \neq \alpha_i\}$, create a new fresh constant d', let $o = d'$ and add the pair (α_i, d') to memory. The second case corresponds to reading an address that has never been written to before in which case a new constant is returned.

f. If g is a memory write operation. This case is similar to a memory read. A diagram is shown in figure 3.

4. Assign to L' the values given to the corresponding next state latch variables in gate graph. Assign to M' the address/values pairs in memory at the end of step 3. P' is defined as above.

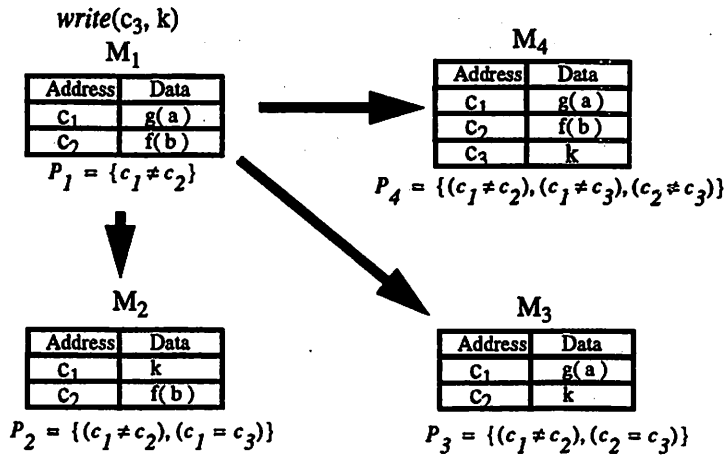
Step 3 is referred to as ***value propagation***. Note that the configuration graph is finite-branching, i.e. for every state, there are a finite number of next states. It is possible that a table is not complete, i.e. there are inputs for which there are no outputs. Then, the set of values assigned to an output of a table may be empty. The empty values propagate, i.e. if one of the inputs to a table is empty, then the output is empty as well.

Now that we have defined an ICS state and the operational semantics we can now define the state space of systems modeled using ICS:

Definition An ***ICS state graph*** G, for an ICS model M, is a directed graph where each node is an ICS state. $(u, v) \in G$ is an edge in the graph if there is a transition between ICS states u and v that is allowable by the operational semantics. Note that the initial state $u_0 = (latch_{init}, \emptyset, \emptyset)$ is a node in the graph (there can be multiple initial states). In addition, only ***reachable*** states are allowed in G (i.e. there must be a path from an initial state, u_0, to each state w in G).

2.3 ICS Models and Verification

ICS models can be used to model and verify hardware systems. Fairness constraints, which rule out some unwanted behavior (introduced due to abstraction), can be placed on finite latches. In order to verify a property, the property is written as an ICS model, and it is checked that the language of the system is contained in the language of the property. A string in the language of an ICS model M is obtained by traversing a path in the graph of the operational model of M, such that the set of infinitely occurring states of the finite latches satisfy the fairness constraints. The language containment must hold for any interpretation given to uninterpreted functions and constants, i.e. for all possible definitions for the functions and valuations for the constants. In some cases, the property can be complemented automatically. In other cases, the user must do this by hand. The complement of the property is then composed with the system, and it is checked that the language of the composed system is empty. Hence, the verification problem reduces to checking whether the language of an ICS model is empty. Again, emptiness must hold for all interpretations given to uninterpreted functions and constants.



In the top left diagram, write(c_3, k) is performed in a state with memory M_1 and predicate P_1. Since c_3 is not equal to the addresses c_1 and c_2 currently stored in M_1, all cases have to be explored: $c_3 = c_1$, $c_3 = c_2$ and $c_1 \neq c_2 \neq c_3$. This creates three new states, along with their associated memories and predicates, M_2, M_3, M_4 and P_2, P_3 P_4.

Figure 3

3 State Reduction Methods

The ICS paradigm only allows finite control variables to be specified explicitly in the property. Thus, any state reduction technique that preserves the sequential behavior of all finite variables and latches will not cause any loss in verification accuracy with respect to the property being verified. We formally define this notion below.

Definition Let $F(s)$ denote the projection of the set of traces from s to the set of finite variables. Two states s_1 and s_2 are said to be ***trace equivalent*** iff $F(s_1) = F(s_2)$.

Theorem 1 Let s_1 and s_2 be two trace equivalent states in a symbolic graph G. Let G' be obtained from G by deleting s_2 and replacing all edges of the form (v, s_2) by (v, s_1) . Then, $L(G) = \emptyset$ iff $L(G') = \emptyset$.

Proof This follows from the definition of trace equivalent states and the fact that fairness constraints and hence language containment only depend on the finite variables (QED).

3.1 Deleting Isomorphic States

In this section, we show that two states may be trace equivalent, even though the ICS terms assigned to the integer latches, predicates and memory may be different. Two states are isomorphic if they have equivalent assignments to finite latches and one can replace the set of constants occurring in one state with a different set of constants such that the result is syntactically equivalent to the second state. An example of such a case is shown in Figure 4.

Definition Let the ***constants of an ICS term*** t, $Cons(t)$, be the set of constants appearing in t. Let the ***constants of a predicate*** p be the union of all constants appearing in the ICS terms of p. Similarly, define the constants of a state and a set of predicates.

Definition Two ICS terms t_1 and t_2 are said to be ***isomorphic*** iff there exists an isomorphism $I: Cons(t_1) \rightarrow Cons(t_2)$ where $I(t_1) = t_2$ (and consequently $t_1 = I^{-1}(t_2)$). Intuitively, two ICS terms are the isomorphic if they are the same up to a renaming of the constants. Similarly, define isomorphic predicates, sets of predicates, and states.

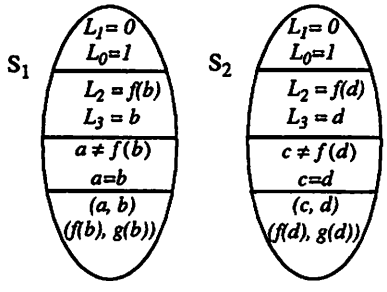
Lemma 3.1 Let two sets of n predicates $P = \{p_1, \ldots, p_n\}$ and $\tilde{P} = \{\tilde{p}_1, \ldots, \tilde{p}_n\}$ be given such that P and $\tilde{P}$ are isomorphic. We have,

1. $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_n$ is valid iff $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is valid;

2. $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_n$ is satisfiable iff $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is satisfiable.

Proof We will show if $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_n$ is valid, $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is also valid. The reverse direction and part 2 are similar. Assume to the contrary that $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is not valid. Let $c_1, \ldots, c_m$ and $\tilde{c}_1, \ldots, \tilde{c}_m$ be the constants in P and $\tilde{P}$ respectively. Let $\tilde{I}$ be an interpretation for $\tilde{c}_1, \ldots, \tilde{c}_m$, and the uninterpreted functions and predicates of $\tilde{P}$, making $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ false. Let I, an interpretation for P, be derived from $\tilde{I}$ by making the same assignments to $c_1, \ldots, c_m$ and the uninterpreted functions and predicates of P as $\tilde{I}$ does to $\tilde{c}_1, \ldots, \tilde{c}_m$ and the uninterpreted functions and predicates of $\tilde{P}$. Note that the uninterpreted functions and predicates of P and $\tilde{P}$ are the same. Since P and $\tilde{P}$ are isomorphic, the values of each p_i and $\tilde{p}_i$ under I and $\tilde{I}$ are the same. It follows that $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_n$ is false under I, which is in contradiction to $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_n$ being valid (QED).

Definition Given a state s, a topological sort of the gate graph O, and a generalized gate g, a ***partial state*** at g, denoted by $P(s, g, O)$, is a state reached during symbolic simulation at gate g when processing the gates at the order given by O. This state is defined by the values of finite and integer latches at s, the values assigned to the outputs of the gates processed so far, and the current values of the predicates and memories. Abusing the notation, we sometimes write $P(s, g)$ for $P(s, g, O)$, knowing that a topological sort O is implied.



States S_1 and S_2 are isomorphic since the constant a can be mapped to c, and b to d. The states are trace equivalent.

Figure 4

Theorem 2 Two isomorphic states are trace equivalent.

Proof Let $R(s_1, s_2)$ hold iff s_1 and s_2 are isomorphic. It suffices to show that R is a bisimulation on the symbolic graph G preserving finite values, i.e. if $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold for some assignment to finite variables a, there exists t_2 such that $R(t_1, t_2)$ and $T(s_2, a, t_2)$ hold, and vice versa. We will show one direction since the reverse is similar. Let $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold. Consider a topological sort O of the gate graph used in obtaining t_1 from s_1. Assume by induction that when gate g is processed, R holds of the partial states $P(s_1, g)$ and $P(s_2, g)$ (i.e. the partial states are isomorphic). It suffices to show that after processing g, R still holds of the partial states. Let P_1 and P_2 be the predicates in $P(s_1, g)$ and $P(s_2, g)$ respectively. Based on g, we have the following cases:

1. g is a finite gate. Since by the inductive assumption the inputs to g are the same, assign the same value to the outputs.
2. g is a constant creator. Have the outputs of g, which are new constants, map to one another.
3. g is an integer function. Just apply the function in both cases. By the inductive assumption, the outputs will also be the same up to a renaming of the constants.
4. g is an integer predicate. By the inductive assumption that P_1 and P_2 are isomorphic, and by lemma 3.1, the output of g is determined by P_1 iff the output of g is determined by P_2 (and these values are the same). If both true and false cases

have to be considered, have the corresponding cases map to one another.

5. g is a read. Let the address being read in $P(s_1, g)$ be α_1. We have the following cases.

a. $P_1 \rightarrow (\alpha_1 = \beta_1)$ is valid for some address β_1. Then by lemma 3.1 $P_2 \rightarrow (I(\alpha_1) = I(\beta_1))$ is also valid. By the inductive assumption, the values read in both states (i.e. the values stored in β_1 and $I(\beta_1)$) are isomorphic.

b. $P_1 \rightarrow (\alpha_1 = \beta_i)$ is not valid for any address β_i, but for some address β_1, we have assumed $\alpha_1 = \beta_1$. We have that $P_1 \wedge (\alpha_1 \neq \beta_2) \wedge \ldots \wedge (\alpha_1 \neq \beta_n) \rightarrow (\alpha_1 = \beta_1)$ is satisfiable, which by lemma 3.1, implies $P_2 \wedge (I(\alpha_1) \neq I(\beta_2)) \wedge \ldots \wedge (I(\alpha_1) \neq I(\beta_n)) \rightarrow (I(\alpha_1) = I(\beta_1))$ is also satisfiable. By $P_1 \rightarrow (\alpha_1 = \beta_i)$ not being valid for any address β_i, and by lemma 3.1, we conclude that for no address γ_i in $P(s_2, g)$, $P_2 \rightarrow (I(\alpha_1) = \gamma_i)$ is valid. It follows that the possibility of $I(\alpha_1) = I(\beta_1)$ will also be considered in $P(s_2, g)$. By the inductive assumption, the values read in both states (i.e. the values stored in β_1 and $I(\beta_1)$) are isomorphic.

c. The address being read in $P(s_1, g)$ is distinct from all other addresses. By lemma 3.1, this possibility also exists in $P(s_2, g)$. In both cases a new location and data value is created. Extend I by mapping these new locations and data values to each another respectively.

6. g is a write. Similar to the case of read (QED).

3.2 Propagating Equalities

In this section, we show that some equality predicates in a state may be removed if all the occurrences of one term in the predicate are replaced with the other. An example of such a case is shown in Figure 5.

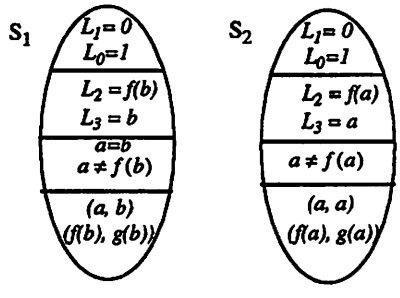
In order to prove that the propagating equalities optimization is valid, the following lemma has to be proven (see the appendix for detail proof).

Lemma 3.2 Let two sets of n predicates $P = \{p_1, \ldots, p_n\}$ and $\tilde{P} = \{\tilde{p}_1, \ldots, \tilde{p}_n\}$ be given such that if all occurrences of some constant b in $p_1, \ldots, p_n$ are replaced by some constant a, then P and $\tilde{P}$ are isomorphic.

a. $p_1 \wedge \ldots \wedge p_{n-1} \wedge (a = b) \rightarrow p_n$ is valid iff $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is valid.

b. $p_1 \wedge \ldots \wedge p_n \wedge (a = b)$ is satisfiable iff $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_n$ is satisfiable.

Definition We say state t is the result of applying the ***equality propagation operator*** to state s if t is isomorphic to some state $\tilde{s}$ obtained from s by the deletion of some equality predicate $a = b$ and renaming all occurrences of b in the ICS terms of s by a. Note that the output of the equality propagation operator is well specified up to isomorphism of states.



State S_2 has been obtained from S_1 by equality propagation. Here all occurrence of b have been replaced by a. The states are trace equivalent.

Figure 5

Theorem 3 If t is the result of applying the equality propagation predicate to s, then s and t are trace equivalent.

3.3 Simplifying Functional Terms

In this section, we prove that some ICS terms in a state can be replaced with a fresh constant, while still maintaining finite trace behavior. We call this technique simplifying functional terms. An examples in given in Figure 6.

Terminology Let $P(\alpha_1, ..., \alpha_k)$ be a predicate. We say γ is a term in $P(\alpha_1, ..., \alpha_k)$ if $\gamma = \alpha_j$ for some α_j. We say γ occurs in $P(\alpha_1, ..., \alpha_k)$ if γ is a subterm in one of the α_j's.

In order to prove that the simplifying functional terms optimization is valid, the following lemma has to be proven.

Lemma 3.3 Let $p_1, ..., p_n$ be such that $p_1 \wedge ... \wedge p_n$ is satisfiable. Let $\beta = f(\alpha_1, ..., a, ..., \alpha_k)$, where f is an uninterpreted function, a is not one of the ICS terms in the p_i's (although a can occur in the p_i's), and if a occurs in a term of the form $g(\gamma_1, ..., a, ..., \gamma_l)$, then g in an uninterpreted function. Let $q_1, ..., q_n$ be isomorphic to a set of predicates $o_1, ..., o_n$ obtained from $p_1, ..., p_n$ by replacing all occurrences of β with a new constant b. Then, the following holds.

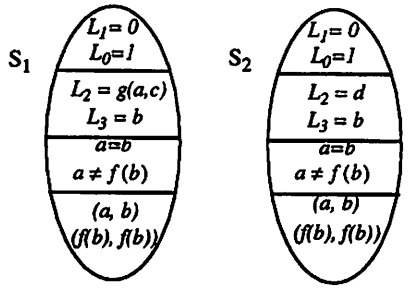
a. $p_1 \wedge ... \wedge p_{n-1} \rightarrow p_n$ is valid iff $q_1 \wedge ... \wedge q_{n-1} \rightarrow q_n$ is valid.

b. $p_1 \wedge ... \wedge p_{n-1} \rightarrow p_n$ is satisfiable iff $q_1 \wedge ... \wedge q_{n-1} \rightarrow q_n$ is satisfiable.

Remark Condition 3 of lemma 3.3 is needed to avoid the following situation. Assume in some state s, the two integer latches contain $f(a)$ and $f(c)$ respectively, and the predicate $a + 1 = c + 1$ holds. Then $f(a)$ should not be replaced by a new constant. Condition 3 ensures that this will not happen.

Notation We say a term α is stored in state s if α is stored in one of the integer latches, or is the address or data in some memory location.

Definition Let a state s and an ICS term $\beta = f(\alpha_1, ..., a, ..., \alpha_n)$ be given. Assume constant a is not one of the ICS terms stored in s, is not a term in any of the predicates of s, and if a occurs in a term of the form $g(\gamma_1, ..., a, ..., \gamma_l)$, then g in an uninterpreted function. We say state t is the result of ***simplifying the functional term*** β in s if t is isomorphic to some state u obtained from s by replacing the term β in all ICS terms of s by a new fresh constant b.



State S_2 has been obtained from S_1 by simplifying functional terms. Here the expression $g(a, c)$ has been replaced with a constant d since c does not occur in any other term in S_1. The states are trace equivalent.

Figure 6

Theorem 4 If t is the result of simplifying a functional term in s, then s and t are trace equivalent.

Remark In ICS models, it is possible for some infinite run to have no interpretations in standard integers ([HIB96]). Assume t is obtained by simplifying a functional term in s. Then, there may be an infinite run r_s from s which has no interpretations in standard integers, but its corresponding run r_t from t has an interpretation in standard integers. For example, let s contain $f(a)$ in some integer latch, and assume a does not appear anywhere else in the system. Then, s is trace equivalent to some state t in which $f(a)$ has been replaced by some new constant b. Assume from s the only possible next state is s_0 in which $f(a) = 0$. Further assume the only possible next state from s_0 is s_1 in which $f(0) = 1$. Proceeding inductively, assume the only next state from s_i is s_{i+1} in which $f(i) = i + 1$. Let

$r_s = s, s_0, s_1, \ldots$. Let $r_t = t, t_0, t_1, \ldots$ be the run corresponding to r_s. We have $f(i) = i+1$ holds in t_i. Run r_s is not satisfiable in standard integers since if $a = n$, then both $f(a) = 0$ and $f(n) = n+1$ should hold. However, r_t is satisfiable by letting $b = 0$.

3.4 Deleting Dangling Predicates

In this section, we prove that a predicate in a state can be removed if the predicate contains a constant that do not occur in any other term in that state. We call this reduction deleting dangling predicates. An example is given in Figure 7.

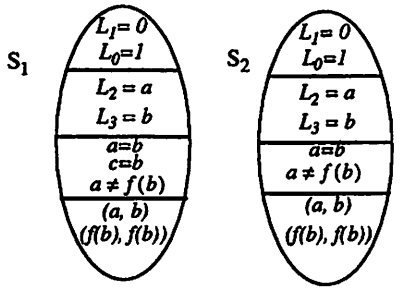
In order to prove that the deleting dangling predicates optimization is valid, the following lemma has to be proven.

Lemma 3.4 Let $n+1$ predicates $p_1, \ldots, p_{n+1}$ be given such that $p_1 \wedge \ldots \wedge p_{n+1}$ is satisfiable. Let p_n be either an uninterpreted predicate $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$, or an equality predicate $c_0 = c_k$, or an inequality predicate $c_0 \neq c_k$ such that c_0 occurs in the other p_i's only in terms of the form $g(\gamma_1, \ldots, c_0, \ldots, \gamma_k)$, where g is an uninterpreted function. Then,

a. $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ is valid iff $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_{n+1}$ is valid.

b. $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ is satisfiable iff $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_{n+1}$ is satisfiable.

Definition A constant c_0 in M is said to be ***absent*** if c_0 appears in the predicates, integer latches, addresses and data values only in terms of the form $g(\gamma_1, \ldots, c_0, \ldots, \gamma_k)$, where g is an uninterpreted function. We assume t is the result of ***deleting a dangling predicate*** from s, if there exists an equality predicate $c_0 \neq c_k$, or an inequality predicate $c_0 = c_k$, or an uninterpreted predicate $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$ in s with c_0 absent in s, such that deletion of this predicate from s results in a state isomorphic with t.



State S_2 has been obtained from S_1 by deleting dangling predicates. Here the predicate $c = d$ has been removed since c does not occur in any other term in S_1. The states are trace equivalent.

Figure 7

Theorem 5 Let t be the result of deleting dangling predicate $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$ from s. Then, s is trace equivalent to t.

3.5 Deleting Dangling Memory Locations

In this section, we prove that a memory location in a state can be removed if the address of the location is a constant that does not occur in any other term in the state. We call this reduction deleting dangling memory locations. An example is given in Figure 8.

Recall that two sets of predicates $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ are equivalent, written as $P \equiv Q$, iff $p_1 \wedge \ldots \wedge p_n \leftrightarrow q_1 \wedge \ldots \wedge q_m$ is valid.

Definition Let P_s denote the predicates of state s. We say two states s and t are ***predicate equivalent*** iff $P_s \equiv P_t$, s and t agree (have the same values) on the finite latches, integer latches, and memory elements.

Notation Let s be a state and p be a predicate. Then s_p denotes the state obtained by adding p to the set of predicates of s, assuming that s_p is well-defined. For example, if $f(a)$ and $f(b)$ are two memory addresses of a memory element,

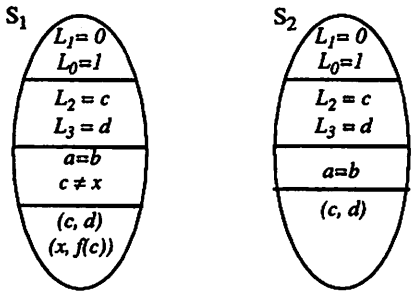
$s_{a=b}$ is undefined, since by the semantics of ICS models, $f(a) \neq f(b)$ in s, and $(a = b) \wedge (f(a) \neq f(b))$ is unsatisfiable.

In order to prove that the deleting dangling memory locations optimization is valid, the following lemmas has to be proven.

Lemma 3.5 Two predicate equivalent states are trace equivalent.

Lemma 3.6 For any state s and predicate p, $F(s_p) \subseteq F(s)$.

Definition Let a be a constant and γ an ICS term. A memory address location (a, γ) in some state s is said to be ***inaccessible*** if a does not appear anywhere else in the system, i.e. not in any of the integer latches, non-address predicates, or memory locations. ***Address predicates*** are inequality predicates of the form $\alpha_1 \neq \alpha_2$, where α_1 and α_2 are memory addresses defined in state s. Intuitively, address predicates are predicates created to ensure that all memory locations are distinct.



State S_2 has been obtained from S_1 by deleting dangling memory locations. Here, the memory location $(x, f(c))$ can be removed since x does not occur in any other ICS term in S_1 (except for the address predicate $c \neq x$). The states are trace equivalent.

Figure 8

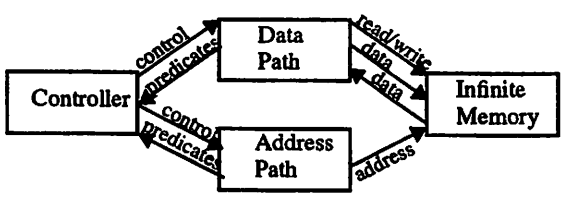
Theorem 6 A state s is trace equivalent to a state t obtained from s by deleting an inaccessible location (a, γ) .

4 Some Implications of Optimized ICS Reachability

In this section, we define a new class of ICS models called decoupled data independent memory systems which, once the optimizations in this paper have been performed, have a finite state graph.

4.1 Decidability of Language Containment in Some Cases

Definition A ***decoupled data independent memory system (DDIMS)*** is an ICS model consisting of four parts: controller, data path, address path, and an infinite memory. The controller is a set of finite variables, and tables on them. A data variable can be the output of data movement elements, constant creators, or read operations. An address variable can be the output of a data movement element or a constant creator. The write operations to memory are of the form $write(a, d)$, where a is an address and d is the data. The only predicates allowed in the data and address paths are equality comparators of the form $x = y$.



The general configuration of a DDIMS. The controller sends control information to the data and address paths. The address and data paths send the results of inspection of data and addresses back to the controller. The data path sends and receives data to and from memory. The address path provides the addresses of various locations to the memory.

Figure 9

We note the following facts about DDIMSs.

1. In DDIMSs data and addresses are separate. Specifically, addresses cannot be stored in memory.
2. DDIMSs can be recognized in polynomial time. Given that, we assume the ICS reachability engine uses different sets of constants for addresses and data variables.
3. Let a_i and a_j denote addresses. Since equalities of the form $a_i = a_j$ are propagated (section 3.2), the only predicates

involving addresses are inequalities of the form $a_i \neq a_j$. Recall that, since all addresses of a memory element are assumed to be distinct, predicates specifying inequality of addresses of memory elements are not part of the set of predicates of a state. Therefore, inequalities of the form $a_i \neq a_j$ must involve some address not in memory. Hence, either a_i or a_j is an address stored in an address latch. It can be shown using the same proof technique as the proof of theorem 7 that in performing reachability of DDIMSs, inequalities of the form $a_i \neq a_j$ can be deleted while preserving language emptiness. We conclude the only predicates in reachability of DDIMSs are inequalities between data variables.

We are now ready to prove that the language emptiness problem for DDIMSs is decidable.

Theorem 7 Let M be a DDIMS. Then, the optimized symbolic simulation algorithm terminates, and generates a graph G such that $L(M) = \emptyset$ iff $L(G) = \emptyset$.

Proof Let d be the number of integer latches in the data path, a the number of integer latches in the address path, and $n = a + d$ the total number of integer latches. Since in DDIMSs no address can be stored in memory, and by the fact that there are no predicates involving addresses in the reachability of DDIMSs, there are at most a accessible memory locations (section 3.5). By theorem 6, no configuration with more than a memory locations is needed. Similarly, there are at most $n = a + d$ accessible data values, d in the data path integer latches, and a in memory. Since equality predicates are propagated (section 3.2), the only predicates in a state are inequalities of the form $d_i \neq d_j$, where d_i and d_j are data constants. Since there can be at most n data constants stored in data latches and memory, dangling predicates are deleted (section 3.4), and isomorphic states are renamed (section 3.1), only n data constants will be ever needed, and there can be at most $O(n^2)$ (data) predicates.

Assume the symbolic simulator uses constants $a_1, \ldots, a_a$ for addresses, and $d_1, \ldots, d_n$ for data, then each data latch or data location in memory can take on one of the d_i's, and an address variable one of the a_i's. It follows that G is finite and of size $O(f 2^{n^2} a^a n^{d+1})$, where f is the number of finite states, 2^{n^2} is due to different predicate configurations, a^a comes from various configurations for address latches, and n^{d+1} is due to different configurations of data latches and memory locations. Since all of our optimizations preserve language emptiness, $L(M) = \emptyset$ iff $L(G) = \emptyset$ (QED).

4.2 Exponential Compaction Compared to Finite Instantiations

Lemma 4.1 There is a DDIMS M with $n+1$ control states, such that $L(M) \neq \emptyset$, the ICS reachability routine generates $n+1$ states, but any finite instantiation with non-empty language has $O(2^n)$ reachable states.

Proof Let the program have one constant creator in its address path, a latch l in its data path, and a comparator between l and some intermediate line x. We assume the initial value of l is the constant d, and M never changes the value of l. In every control state i, $0 \leq i \leq n-1$, M creates a new constant a_i in its address path, reads the value d_i stored at a_i into x, and compares x to l (which contains d). If $d_i \neq d$, M writes d into a_i, and goes to state $i+1$. Otherwise, M stops (i.e does not accept). State n is a dead accepting state. The only way M can have an infinite path is by constantly reading a new location in memory, since all locations which have been visited before have the value d stored in them. Since the address of the new location is not stored anywhere, the optimized ICS reachability routine always drops the new memory location (section 3.5), and therefore, explores only $n+1$ states. Any finite instantiation preserving the infinite trace must have at least n memory locations, and 2 data values. Since all assignment of values to memory locations is reachable, the finite instantiation contains $O(2^n)$ states (QED).

Remark Even if we assume that there is only one initial state where all memory locations take the value 0, all possible

assignments of values to the memory locations is reachable, which implies the number of reached states is $O(2^n)$ (assuming 2 data values). If a tool such as Murphi ([DDHY92]) with interleaving semantics, explicit state enumeration, and symmetry reduction is used, the number of explored states is $n+1$, whereas the running time is $O(n^2)$, since at every state $O(n)$ choices have to be considered. If a BDD-based tool is used, the running time is still quadratic. The reason is that the set of memory configurations reached after i steps is all assignments of values to memory locations with i 1's (and $n-i$ 0's). For $i = n/2$, the BDD representing the set of reachable states is of size $O(n^2)$.

5 Experimental Results

We have implemented some of the optimizations presented in this paper by extending the symbolic reachability analyzer described in [IHB96]. These optimizations include performing equality propagation, deleting dangling predicates and deleting dangling memory locations. Our implementation is unable, however, to detect all isomorphic states. We leave efficient detection of such states as an open research problem.

Table 1 shows the results of our experiments with the example given in proof of lemma 4.1. The columns are the number of control states in the example, the number of states reached by ICS reachability, the time taken by VIS (Berkeley's second generation BDD-based verifier, [VIS96]) and the number of states reached by VIS. All experiments were run on a DEC-Alpha server 21164 running at 250MHz with 1 GB of main memory. The reported times are in CPU seconds. As one may notice, the time for ICS reachability grows more than linearly. The reason is that in our implementation, a hybrid implicit/ explicit approach was used, where BDDs are used to represent the control portion of the model, and a special type of symbolic simulation generates the behavior of the datapath (see [IHB96] and [HB95] for a more detailed discussion). In this particular example, the size and complexity of the transition relation grew with the number of control states.

The ICS reachability algorithm can also be implemented explicitly without using BDDs, in which case, we expect linear run times for this example. Despite this, however, VIS was only able to complete the experiments with 4, 8 and 16 control states, and ran out of memory for all the other examples. In addition, the time that VIS took to complete each examples was significantly slower than ICS Reachability. As discussed in the proof lemma 4.1, VIS requires at least n latches in the datapath to model the memory of a DDIMS with $n+1$ control states. The number of latches for memory does not grow when ICS reachability is performed, since dangling memory locations were deleted.

Table 1

Control State Num	4	8	16	32	64	128	256	512	1024	2048	4096	8192
ICS, Time	0.0	0.0	0.1	0.2	0.3	0.7	1.8	4.9	14.7	48.6	173.7	658.3
ICS, Reached States Num	4	8	16	32	64	128	256	512	1024	2048	4096	8192
VIS, Time	0.1	1.2	883.9	-	-	-	-	-	-	-	-	-
VIS, Reached States Num	114	3050	1310000	-	-	-	-	-	-	-	-	-

6 Conclusions and Future Work

This paper presented a set of optimization routines which can drastically reduce the sizes of state spaces of ICS models. Specifically, they can reduce infinite state spaces into finite ones. These results lead to a decision procedure for language emptiness of decoupled data insensitive memory systems, and to show that ICS reachability can have a computational advantage (at least asymptotically) with respect to finite instantiations and the best known algorithms for generating their state spaces.

We see two future research directions.

1. Extend the optimization routines, and hopefully get decision procedures for more circuit families. Two specific open problems where one may be able to get decision procedures (that is if the problems are decidable) using more powerful optimization routines are:

a. the language emptiness problem for data insensitive memory systems, in which data and addresses are not separate, and therefore addresses can be stored in memory;

b. the problem of proving the property "when b becomes true, $x = y$"where x and y are integer variables, and the system is a data computation controller in which the datapath contains data movement operations and uninterpreted functions, but no predicates ([HIKB96]).

2. Gain experience with the impact of these optimizations in practice. A good example may be the verification of the ATM switch described in [LTZSC96].

Acknowledgments

During this work, the first and second authors were supported by SRC grant 96-DC-324.

References

[BD94] J. Burch, D. Dill, *"Automated Verification of Pipelined Micro-processors"*, Computer-Aided Verification, 1994.

[Bry86] R. E. Bryant. *"Graph-based Algorithms for Boolean Function Manipulation."* IEEE Transactions on Computers, 35(9):677-691, August 1986.

[Cor93] F. Corella, *"Automated High-Level Verification Against Clocked Algorithmic Specifications"*, Proceedings of the IFIP WG10.2 Conference on Computer Hardware Description Languages and their Applications, Ottawa, Canada, Apr. 1993. Elsevier Science Publishers B.V.

[DDHY92] D. Dill, A. J. Drexler, A. J. Hu, C. H. Yang, *"Protocol Verification as a Hardware Design Aid"*, International Conference on Computer Design (ICCD) 1992.

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

[HIKB96] R. Hojati, A. Isles, D. Kirkpatrick, R. K. Brayton, *"Verification Using Uninterpreted Functions and Finite Instantiations"*, Formal Methods in Computer-Aided Design, November 1996.

[HIB96] Ramin Hojati, Adrian Isles, Robert K. Brayton, *"ICS Models with Bounded Integers"*, unpublished manuscript, May 1996.

[IHB96] A. Isles, R. Hojati, R. K. Brayton, *"Reachability Analysis of ICS Models"*, SRC Techcon, September 1996.

[LTZSC96] M. Langevin, S. Tahar, Z. Zhou, X. Song, E. Cerny, *"Behavioral Verification of an ATM Switch Fabric using Implicit Abstract State Enumeration"*, International Conference on Computer Design: VLSI in Computers and Processors, Austin, October 1996.

[SDB96] J. X. Su, D. Dill, C. W. Barrett, *"Automatic Generation of Invariants in Processor Verification"*, Formal Methods in Computer-Aided Design, November 1996.

[Sho79] R.E. Shostack, *"A Practical Decision Procedure for Arithmetic With Function Symbols"*, JACM Volume 26, No. 2, April 1979, pp 351-360.

[VIS96] *"VIS: A system for Verification and Synthesis"*, The VIS Group. In the Proceedings of the 8th International Conference on Computer Aided Verification, p428-432, Springer Lecture Notes in Computer Science, #1102, Edited by R. Alur and T. Henzinger, New Brunswick, NJ, July 1996.

Appendix

Lemma 3.2 Let two sets of n predicates $P = \{p_1, ..., p_n\}$ and $\tilde{P} = \{\tilde{p}_1, ..., \tilde{p}_n\}$ be given such that if all occurrences of some constant b in $p_1, ..., p_n$ are replaced by some constant a, then P and $\tilde{P}$ are isomorphic.

a. $p_1 \wedge ... \wedge p_{n-1} \wedge (a = b) \rightarrow p_n$ is valid iff $\tilde{p}_1 \wedge ... \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is valid.

b. $p_1 \wedge ... \wedge p_n \wedge (a = b)$ is satisfiable iff $\tilde{p}_1 \wedge ... \wedge \tilde{p}_n$ is satisfiable.

Proof We will show that if $p_1 \wedge ... \wedge p_{n-1} \wedge (a = b) \rightarrow p_n$ is valid, then $\tilde{p}_1 \wedge ... \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is also valid. The reverse direction of part a, and part b are similar. Assume to the contrary that $\tilde{p}_1 \wedge ... \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is not valid. Let $a, b, c_1, ..., c_m$ and $\tilde{a}, \tilde{c}_1, ..., \tilde{c}_m$ be the constants in P and $\tilde{P}$ respectively. Let $\tilde{I}$ be an interpretation for $\tilde{P}$, making $\tilde{p}_1 \wedge ... \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ false. Let I, an interpretation for P, be derived from $\tilde{I}$ by making the same assignments to

$a, c_1, \ldots, c_m$ and the uninterpreted functions and predicates of P as $\tilde{I}$ does to $\tilde{a}, \tilde{c}_1, \ldots, \tilde{c}_m$ and the uninterpreted functions and predicates of $\tilde{P}$. Let $I(b) = I(a) = \tilde{I}(\tilde{a})$. For each i, if p_i does not contain b, then it is isomorphic to $\tilde{p}_i$, and hence $I(p_i) = \tilde{I}(\tilde{p}_i)$. If p_i contains b, then $\tilde{p}_i$ contains $\tilde{a}$ in its place. Since $I(b) = \tilde{I}(\tilde{a})$, it follows again that $I(p_i) = \tilde{I}(\tilde{p}_i)$. Since $\tilde{p}_1 \wedge \ldots \wedge \tilde{p}_{n-1} \rightarrow \tilde{p}_n$ is false under $\tilde{I}$, and since $I(p_i) = \tilde{I}(\tilde{p}_i)$ for all i, we have that $p_1 \wedge \ldots \wedge p_{n-1}$ is true and p_n is false under I. Since $a = b$ is true under I, we conclude that $p_1 \wedge \ldots \wedge p_{n-1} \wedge (a = b) \rightarrow p_n$ is false under I, which is in contradiction to $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_n$ being valid (QED).

Theorem 3 If t is the result of applying the equality propagation predicate to s, then s and t are trace equivalent.

Proof Let $R(s_1, s_2)$ hold iff s_2 can be obtained by applying the equality propagation operator to s_1. It suffices to show that R is a bisimulation on the symbolic graph G preserving finite values. Let $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold, where a is an assignment of values to the finite variables. We need to show that there exists t_2 such that $T(s_2, a, t_2)$ and $R(t_1, t_2)$ hold (the reverse direction is similar). Further assume that s_1 contains a predicate $a = b$ which was deleted by the equality propagation operator, and all b 's were renamed to a 's. Proceeding by induction, assume that before processing a gate g, R holds of the partial states $P(s_1, g)$ and $P(t_1, g)$.

We define the following notations for the rest of the proof. Let P_1 and P_2 be the predicates in $P(s_1, g)$ and $P(s_2, g)$ respectively. Let I be an isomorphism which maps $P(s_1, g)$ after b has been replaced by a to $P(s_2, g)$. Let $I_{b \rightarrow a}(\alpha)$ be the mapping which first renames b 's to a 's in the ICS term α and then applies I to the result. Similarly define $I_{b \rightarrow a}$ applied to a state. Let the two new partial states after processing g be u_1 and u_2. Based on g, we have the following cases:

1. g is a finite gate. Since by the inductive assumption the inputs to g are the same, assign the same value to the outputs.
2. g is a constant creator. Have the outputs of g map to one another. Extend I with this mapping.
3. g is an integer function. Just apply the function in both cases. We have $I_{b \rightarrow a}(u_1) = u_2$.
4. g is an integer predicate. By the inductive assumption and by lemma 3.2, we have that the output of g is determined by P_1 iff the output of g is determined by P_2, and they are the same. If both true and false cases have to be considered, have the corresponding cases map to one another.
5. g is a read. Let the address being read in $P(s_1, g)$ be α_1. We have the following cases.

5a. $P_1 \rightarrow (\alpha_1 = \beta_1)$ is valid for some address β_1. By lemma 3.2, $P_2 \rightarrow (I_{b \rightarrow a}(\alpha_1) = I_{b \rightarrow a}(\beta_1))$ is also valid. By the inductive assumption, the data value at α_1 is mapped by $I_{b \rightarrow a}$ to the data value at $I_{b \rightarrow a}(\alpha_1)$.

5b. $P_1 \rightarrow (\alpha_1 \neq \beta_i)$ is valid for all addresses β_i. By lemma 3.2, $P_2 \rightarrow (I_{b \rightarrow a}(\alpha_1) \neq I_{b \rightarrow a}(\beta_1))$ is also valid for all β_i. In both cases a new location and data value is created. Extend $I_{b \rightarrow a}$ by mapping these locations and data values to each other respectively.

5c. For $\gamma_i \in \{\gamma_1, \ldots, \gamma_k\}$, neither $P_1 \rightarrow (\alpha_1 = \gamma_i)$ nor $P_1 \rightarrow (\alpha_1 \neq \gamma_i)$ holds, but we have assumed $\alpha_1 = \gamma_j$ for some γ_j. Hence, $P_1 \wedge (\alpha_1 = \gamma_j) \wedge \prod_{i \neq j} (\alpha_1 \neq \gamma_i)$ is satisfiable. By lemma 3.2, $P_2 \wedge (I_{b \rightarrow a}(\alpha_1) = I_{b \rightarrow a}(\gamma_j)) \wedge \prod_{i \neq j} (I_{b \rightarrow a}(\alpha_1) \neq I_{b \rightarrow a}(\gamma_i))$ is also satisfiable, which implies we can assume

$I_{b\to a}(\alpha_1) = I_{b\to a}(\gamma_j)$ holds in $P(s_2, g)$. By the inductive assumption, the data value at α_1 is mapped by $I_{b\to a}$ to the data value at $I_{b\to a}(\alpha_1)$.

5d. For $\gamma_i \in \{\gamma_1, \ldots, \gamma_k\}$, neither $P_1 \to (\alpha_1 = \gamma_i)$ nor $P_1 \to (\alpha_1 \neq \gamma_i)$ holds, but we have assumed α_1 is distinct from all other addresses. Hence, $P_1 \wedge \prod_i(\alpha_1 \neq \gamma_i)$ is satisfiable. By lemma 3.2, $P_2 \wedge \prod_i(I_{b\to a}(\alpha_1) \neq I_{b\to a}(\gamma_i))$ is also satisfiable, which implies we can assume $I_{b\to a}(\alpha_1)$ is distinct in $P(s_2, g)$ as well. In both cases a new location and data value is created. Extend $I_{b\to a}$ by mapping these locations and data values to each other respectively.

6. g is a write. Similar to the case of read (QED).

Lemma 3.3 Let $p_1, \ldots, p_n$ be such that $p_1 \wedge \ldots \wedge p_n$ is satisfiable. Let $\beta = f(\alpha_1, \ldots, a, \ldots, \alpha_k)$, where f is an uninterpreted function, a is not one of the ICS terms in the p_i's (although a can occur in the p_i's), and if a occurs in a term of the form $g(\gamma_1, \ldots, a, \ldots, \gamma_l)$, then g in an uninterpreted function. Let $q_1, \ldots, q_n$ be isomorphic to a set of predicates $o_1, \ldots, o_n$ obtained from $p_1, \ldots, p_n$ by replacing all occurrences of β with a new constant b. Then, the following holds.

a. $p_1 \wedge \ldots \wedge p_{n-1} \to p_n$ is valid iff $q_1 \wedge \ldots \wedge q_{n-1} \to q_n$ is valid.

b. $p_1 \wedge \ldots \wedge p_{n-1} \to p_n$ is satisfiable iff $q_1 \wedge \ldots \wedge q_{n-1} \to q_n$ is satisfiable.

Proof We will prove part a; part b is similar. By lemma 3.1, it suffices to prove the lemma assuming that $q_1, \ldots, q_n$ is obtained from $p_1, \ldots, p_n$ by replacing all occurrences of β with b. It suffices to show for all interpretations I for $p_1, \ldots, p_n$, there exists an interpretation J for $q_1, \ldots, q_n$ such that $I(p_i) = J(q_i)$ for all i, and vice versa. Let $M_{\beta\to b}(\alpha)$ be the mapping which replaces all occurrences of β with b in the term α. Given I, let J be obtained from I by setting $J(M_{\beta\to b}(\alpha)) = I(\alpha)$ (specifically $J(b) = I(\beta)$). It follows that $I(p_i) = J(q_i)$ for all i. Conversely, given J, let $I(a)$ be a value not equal to any of the values $J(\gamma)$ for all ICS terms γ occurring in $q_1, \ldots, q_n$. Obtain I from J by letting $I(\beta) = J(b)$, and $I(\alpha) = J(M_{\beta\to b}(\alpha))$ for $\alpha \neq \beta$. Condition 3 in the statement of the lemma guarantees that this can be done. It again follows that $I(p_i) = J(q_i)$ for all i (QED).

Theorem 4 If t is the result of simplifying a functional term in s, then s and t are trace equivalent.

Proof Let $R(s_1, s_2)$ hold iff s_2 is the result of simplifying a functional term in s_1. It suffices to show that R is a bisimulation on the symbolic graph G preserving finite values. Let $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold, where a is an assignment of values to the finite variables. We need to show there exists t_2 such that $R(t_1, t_2)$ and $T(s_2, a, t_2)$ hold. To complete the proof, we need to show the reverse, which is similar and will not be given here. Proceeding by induction, assume that before processing a gate g, the partial state $P(s_1, g)$ contains an ICS term $\beta = f(\alpha_1, \ldots, a, \ldots, \alpha_n)$ which was replaced by a new constant b to obtain $P(s_2, g)$.

We define the following notations for the rest of the proof. Let P_1 and P_2 be the predicates in $P(s_1, g)$ and $P(s_2, g)$ respectively. Let I be an isomorphism which maps $P(s_1, g)$ after β has been replaced by b to $P(t_1, g)$. Let $I_{\beta\to b}(\alpha)$, for a term α, be the mapping which first replaces β's by b's in α and then applies I to the result. Similarly define $I_{\beta\to b}$

applied to a state of an ICS model. Let the two new partial states after processing g be u_1 and u_2. We have the following cases:

1. g is a finite gate. Since by the inductive assumption the inputs to g are the same, assign the same value to the outputs.
2. g is a constant creator. Have the outputs of g map to one another. Extend I by this mapping.
3. g is an integer function. Just apply the function in both cases. We have $I_{\beta \to b}(u_1) = u_2$. The requirement that a occurs only in terms of the form $g(\gamma_1, ..., a, ..., \gamma_k)$ will not be violated in u_1 after processing g.
4. g is an integer predicate. By the inductive assumption and by lemma 3.3, we have that the output of g is determined by P_1 iff the output of g is determined by P_2, and they are the same. If both true and false cases have to be considered, have the corresponding cases map to one another. The requirement that a is not one of the terms in any predicates will not violated since no input of g is a.
5. g is a read. Let the address being read in $P(s_1, g)$ be α_1. Note that $\alpha_1 \neq a$, and for all addresses β_i, $\beta_i \neq a$. We have the following cases.

5a. $P_1 \to (\alpha_1 = \beta_1)$ is valid for some address β_1. By lemma 3.3, $P_2 \to (I_{\beta \to b}(\alpha_1) = I_{\beta \to b}(\beta_1))$ is also valid. By the inductive assumption, the data value at α_1 is mapped by $I_{\beta \to b}$ to the data value at $I_{\beta \to b}(\alpha_1)$.

5b. $P_1 \to (\alpha_1 \neq \beta_i)$ is valid for all addresses β_i. By lemma 3.3, $P_2 \to (I_{\beta \to b}(\alpha_1) \neq I_{\beta \to b}(\beta_1))$ is also valid for all β_i. In both cases a new location and data value is created. Extend $I_{\beta \to b}$ by mapping these locations and data values to each other respectively.

5c. For $\gamma_i \in \{\gamma_1, ..., \gamma_k\}$, neither $P_1 \to (\alpha_1 = \gamma_i)$ nor $P_1 \to (\alpha_1 \neq \gamma_i)$ holds, but we have assumed $\alpha_1 = \gamma_j$ for some γ_j (the requirement that a is not one of the terms in any predicates in u_1 will not be violated since $\gamma_i \neq a$). Hence, $P_1 \wedge (\alpha_1 = \gamma_j) \wedge \prod_{i \neq j} (\alpha_1 \neq \gamma_i)$ is satisfiable. By lemma 3.3, $P_2 \wedge (I_{\beta \to b}(\alpha_1) = I_{\beta \to b}(\gamma_j)) \wedge \prod_{i \neq j} (I_{\beta \to b}(\alpha_1) \neq I_{\beta \to b}(\gamma_i))$ is also satisfiable, which implies we can assume $I_{\beta \to b}(\alpha_1) = I_{\beta \to b}(\gamma_j)$ holds in $P(s_2, g)$. By the inductive assumption, the data value at α_1 is mapped by $I_{\beta \to b}$ to the data value at $I_{\beta \to b}(\alpha_1)$.

5d. For $\gamma_i \in \{\gamma_1, ..., \gamma_k\}$, neither $P_1 \to (\alpha_1 = \gamma_i)$ nor $P_1 \to (\alpha_1 \neq \gamma_i)$ holds, but we have assumed α_1 is distinct from all other addresses. Hence, $P_1 \wedge \prod_i (\alpha_1 \neq \gamma_i)$ is satisfiable. By lemma 3.3, $P_2 \wedge \prod_i (I_{\beta \to b}(\alpha_1) \neq I_{\beta \to b}(\gamma_i))$ is also satisfiable, which implies we can assume $I_{\beta \to b}(\alpha_1)$ is distinct in $P(s_2, g)$ as well. In both cases a new location and data value is created. Extend $I_{\beta \to b}$ by mapping these locations and data values to each other respectively.

6. g is a write. Similar to the case of read (QED).

Lemma 3.4 Let $n+1$ predicates $p_1, ..., p_{n+1}$ be given such that $p_1 \wedge ... \wedge p_{n+1}$ is satisfiable. Let p_n be either an uninterpreted predicate $p(\alpha_1, ..., c_0, ..., \alpha_m)$, or an equality predicate $c_0 = c_k$, or an inequality predicate $c_0 \neq c_k$ such that c_0 occurs in the other p_i's only in terms of the form $g(\gamma_1, ..., c_0, ..., \gamma_k)$, where g is an uninterpreted function. Then,

a. $p_1 \wedge ... \wedge p_n \to p_{n+1}$ is valid iff $p_1 \wedge ... \wedge p_{n-1} \to p_{n+1}$ is valid.

b. $p_1 \wedge ... \wedge p_n \to p_{n+1}$ is satisfiable iff $p_1 \wedge ... \wedge p_{n-1} \to p_{n+1}$ is satisfiable.

Proof. By the definition of validity, $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_{n+1}$ being valid implies that $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ is valid. We will show the reverse; part b is similar. Let $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ be valid. We need to show $p_1 \wedge \ldots \wedge p_{n-1} \rightarrow p_{n+1}$ is valid. Assume to the contrary that there exists an interpretation J such that $J(p_1 \wedge \ldots \wedge p_{n-1}) = T$ and $J(p_{n+1}) = F$. We will show, contrary to the assumption, that there exists an interpretation I such that $I(p_1 \wedge \ldots \wedge p_n) = T$ and $I(p_{n+1}) = F$.

1. If p_n is an uninterpreted predicate $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$, let $I(c_0)$ be a new value not equal to the value of any other term or subterm under J. For every term $g(\gamma_1, \ldots, c_0, \ldots, \gamma_k)$, let $I(g(\gamma_1, \ldots, c_0, \ldots, \gamma_k)) = J(g(\gamma_1, \ldots, c_0, \ldots, \gamma_k))$. This is possible since g in uninterpreted and $I(c_0)$ is not equal to the value of any other term and subterm under J. Let $I(p(\alpha_1, \ldots, c_0, \ldots, \alpha_m))$ be true, and let I agree with J on all other constants, uninterpreted functions and predicates. It follows that $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ is false.

2. If p_n is an equality predicate $c_0 = c_k$, and if J does not assign a value to c_0 or c_k, let I be an extension of J with the extra assignment $I(c_0) = I(c_k)$. It follows that $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ is false under I. If J assigns unequal values to c_0 and c_k, let $I(c_0) = J(c_k) = l$, where l is a new value not equal to the value of any other term and subterm under J. Let I agree with J on all other constants, uninterpreted functions and predicates. Again, $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$ is false under I.

3. Similar to case 2 (if $J(c_0) = J(c_k)$, then let $I(c_o) \neq I(c_k)$ with $I(c_0)$ and $I(c_k)$ taking new values not equal to the value of any other term and subterm under J) (QED).

Theorem 5 Let t be the result of deleting dangling predicate $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$ from s. Then, s is trace equivalent to t.

Proof Let $R(s_1, s_2)$ hold iff s_2 is the result of deleting dangling predicate $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$ from s_1. It suffices to show that R is a bisimulation on the symbolic graph G preserving finite values. Let $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold, where a is an assignment of values to the finite variables. We need to show there exists t_2 such that $R(t_1, t_2)$ and $T(s_2, a, t_2)$ hold. Fix a topological sort of the gate graph. Proceeding by induction, assume that before processing a gate g, the partial state $R(P(s_1, g), P(s_2, g))$ holds. Let the two new partial states after processing g be u_1 and u_2. We will show that after processing g, $R(u_1, u_2)$ holds.

We define the following notations for the rest of the proof. Let P_1 and P_2 be the predicates in $P(s_1, g)$ and $P(s_2, g)$ respectively. Let I be an isomorphism which maps $P(s_1, g)$ after $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$ has been deleted to $P(t_1, g)$. Let $I_p(s)$ be the mapping which first deletes $p(\alpha_1, \ldots, c_0, \ldots, \alpha_m)$ from s, and then applies I to the result. Based on g, we have the following cases:

1. g is a finite gate. Since by the inductive assumption the inputs to g are the same, assign the same value to the outputs.

2. g is a constant creator. Have the outputs of g map to one another, as an extension of the current isomorphism I. Note that by the semantics of ICS models, this new constant in u_1 is not c_0 since c_0 occurs in $P(s_1, g)$.

3. g is an integer function. Just apply the function in both cases. We have $I_p(u_1) = u_2$.

4. g is an integer predicate. By the inductive assumption and by lemma 3.4, we have that the output of g is determined by

P_1 iff the output of g is determined by P_2, and they are the same. If both true and false cases have to be considered, have the corresponding cases map to one another, as far as bisimulation is concerned.

5. g is a read. Let the address being read in $P(s_1, g)$ be α_1. Note that c_0 only occurs in terms of the form $g(\tau_1, \ldots, c_0, \ldots, \tau_k)$.

5a. $P_1 \rightarrow (\alpha_1 = \beta_1)$ is valid for some address β_1. By lemma 3.4, $P_2 \rightarrow (I(\alpha_1) = I(\beta_1))$ is also valid. By the inductive assumption, the data value at α_1 is mapped by I to the data value at $I(\alpha_1)$.

5b. $P_1 \rightarrow (\alpha_1 \neq \beta_i)$ is valid for all addresses β_i. By lemma 3.4, $P_2 \rightarrow (I(\alpha_1) \neq I(\beta_1))$ is also valid for all β_i. In both cases a new location and data value is created. Extend I by mapping these locations and data values to each other respectively.

5c. For $\gamma_i \in \{\gamma_1, \ldots, \gamma_k\}$, neither $P_1 \rightarrow (\alpha_1 = \gamma_i)$ nor $P_1 \rightarrow (\alpha_1 \neq \gamma_i)$ holds, but we have assumed $\alpha_1 = \gamma_j$ for some γ_j. Hence, $P_1 \wedge (\alpha_1 = \gamma_j) \wedge \prod_{i \neq j} (\alpha_1 \neq \gamma_i)$ is satisfiable. By lemma 3.4, $P_2 \wedge (I(\alpha_1) = I(\gamma_j)) \wedge \prod_{i \neq j} (I(\alpha_1) \neq I(\gamma_i))$ is also satisfiable, which implies we can assume $I(\alpha_1) = I(\gamma_j)$ holds in $P(s_2, g)$. By the inductive assumption, the data value at α_1 is mapped by I to the data value at $I(\alpha_1)$.

5d. For $\gamma_i \in \{\gamma_1, \ldots, \gamma_k\}$, neither $P_1 \rightarrow (\alpha_1 = \gamma_i)$ nor $P_1 \rightarrow (\alpha_1 \neq \gamma_i)$ holds, but we have assumed α_1 is distinct from all other addresses. Hence, $P_1 \wedge \prod_i (\alpha_1 \neq \gamma_i)$ is satisfiable. By lemma 3.4, $P_2 \wedge \prod_i (I(\alpha_1) \neq I(\gamma_i))$ is also satisfiable, which implies we can assume $I(\alpha_1)$ is distinct in $P(s_2, g)$ as well. In both cases a new location and data value is created. Extend I by mapping these locations and data values to each other respectively.

6. g is a write. Similar to the case of read (QED).

Lemma 3.5 Two predicate equivalent states are trace equivalent.

Proof The proof uses proposition 1, and is very similar to the inductive proofs we have been doing throughout, and will continue to do (QED).

Proposition 1 If $P \equiv Q$, then $P \rightarrow r$ is valid iff $Q \rightarrow r$ is valid.

Proof We will show validity of $P \rightarrow r$ implies validity of $Q \rightarrow r$. The other direction is similar. To show, $Q \rightarrow r$ is valid, assume Q is true. This implies, by $P \equiv Q$, that P is true, which in turn by the validity of $P \rightarrow r$ implies r is true (QED).

Lemma 3.6 For any state s and predicate q, $F(s_p) \subseteq F(s)$.

Proof The intuitive idea is that by adding more assumptions (i.e predicates), the behavior becomes more restricted. Let $R(s_1, s_2)$ hold iff there exists an isomorphism $I: s_1 \rightarrow s_2$ such that $I(s_1)$ and s_2 agree on finite latches, integer latches, and memory elements, and there is a predicate q such that $I(P_{s_1}) \equiv P_{s_2} \cup q$. Let $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold, where a is an assignment of values to the finite variables. We need to show there exists t_2 such that $T(s_2, a, t_2)$ and $R(t_1, t_2)$ hold, or t_1 and t_2 are trace equivalent. Proceeding by induction, assume that before processing a gate g, $R(P(s_1, g), P(s_2, g))$ holds. Let P_1 and P_2 denote the predicates at $P(s_1, g)$ and $P(s_2, g)$, respectively. Let u_1 and u_2 denote the states obtained after processing gate g in $P(s_1, g)$ and $P(s_2, g)$ respectively. Based on g, we have the following cases:

1. g is a finite gate. Since by the inductive assumption the inputs to g are the same, assign the same value to the outputs.

2. g is a constant creator. Extend I by having the new constants map to each other.

3. g is an integer function. Just apply the function in both cases.

4. g is an integer predicate. Let r_1 and r_2 be the predicates associated with gate g in $P(s_1, g)$ and $P(s_2, g)$ respectively. Several possibilities exist:

a. $P_1 \rightarrow r_1$ is valid. Since by proposition 2, $P_2 \rightarrow \overline{r_2}$ is not valid, we have the following possibilities:

a1. $P_2 \rightarrow r_2$ is valid. Note that by the operational semantics of ICS models u_1 and u_2 do not contain r_1 and r_2, since they are implied by P_1 and P_2. Hence, by the inductive assumption, $R(u_1, u_2)$ holds.

a2. Neither $P_2 \rightarrow r_2$ nor $P_2 \rightarrow \overline{r_2}$ is valid. By proposition 3, $P(s_1, g)$ is related by R to $P(s_2, g)$ augmented with r_2.

b. $P_1 \rightarrow \bar{r}$ is valid. Similar to case a with the roles r_1 and $\overline{r_2}$ switched.

c. Neither $P_1 \rightarrow r_1$ nor $P_1 \rightarrow \overline{r_1}$ is valid. By proposition 4, neither $P_2 \rightarrow r_2$ nor $P_2 \rightarrow \overline{r_2}$ is valid. By proposition 5, $P(s_1, g)$ augmented with r_1 and $P(s_2, g)$ augmented with r_2 are related by R, as are $P(s_1, g)$ augmented by $\overline{r_1}$ and $P(s_2, g)$ augmented with $\overline{r_2}$.

5. g is a read. Let the address being read in $P(s_1, g)$, be α_1.

a. $P_1 \rightarrow (\alpha_1 = \beta_1)$ is valid for some address β_1 in memory. Let r_1 and r_2 be the predicates $\alpha_1 = \beta_1$ and $I(\alpha_1) = I(\beta_1)$, and the values read at β_1 and β_2 be γ_1 and γ_2. Since by proposition 2, $P_2 \rightarrow \bar{r}$ is not valid, we have the following possibilities:

a1. $P_2 \rightarrow r_2$ is valid. Note that by the operational semantics of ICS models u_1 and u_2 do not contain r_1 and r_2, since they are implied by P_1 and P_2). Hence, by the inductive assumption, $R(u_1, u_2)$ holds. By the inductive assumption $I(\gamma_1) = \gamma_2$.

a2. Neither $P_2 \rightarrow r_2$ nor $P_2 \rightarrow \overline{r_2}$ is valid. By proposition 3, $P(s_1, g)$ is related by R to $P(s_2, g)$ augmented with r_2. By the inductive assumption $I(\gamma_1) = \gamma_2$.

b. $P_1 \rightarrow (\alpha_1 \neq \beta_1)$ is valid for some address β_1 in memory. Since by proposition 2, $P_2 \rightarrow (I(\alpha_1) = I(\beta_1))$ is not valid for any address β_1, we have the following possibilities:

b1. $P_2 \rightarrow (I(\alpha_1) \neq I(\beta_1))$ is valid for all β_1. Create a new address and data value in each state, and extend I by mapping the new addresses and data values to each other respectively. Since no new predicates have been created, $R(u_1, u_2)$ still holds.

b2. For some set of addresses $\{\beta_i\}$, neither $P_2 \rightarrow (I(\alpha_1) = I(\beta_i))$ nor $P_2 \rightarrow (I(\alpha_1) \neq I(\beta_i))$ is valid. Consider the case where we have assumed $I(\alpha_1) \neq I(\beta_i)$ for all i, i.e. $I(\alpha_1)$ is distinct from all other memory addresses. Create a new address and data value in each state, and extend I by mapping the new addresses and data values to each other respectively. By repeated application of proposition 3, it follows that $R(u_1, u_2)$ holds.

c. For some address β_i, neither $P_1 \rightarrow \alpha_1 = \beta_i$ nor $P_1 \rightarrow \alpha_1 \neq \beta_i$ is valid, but we have assumed $\alpha_1 = \beta_1$. Let r_1 and r_2 be the predicates $\alpha_1 = \beta_i$ and $I(\alpha_1) = I(\beta_i)$, and the values read at β_1 and β_2 be γ_1 and γ_2. By proposition 4,

neither $P_2 \rightarrow r_2$ nor $P_2 \rightarrow \bar{r_2}$ is valid. By proposition 5, $P(s_1, g)$ augmented with r_1 and $P(s_2, g)$ augmented with r_2 are related by R. By the inductive assumption $I(\gamma_1) = \gamma_2$.

d. For some set of addresses $\{\beta_i\}$, neither $P_1 \rightarrow \alpha_1 = \beta_i$ nor $P_1 \rightarrow \alpha_1 \neq \beta_i$ is valid, but we have assumed $\alpha_1 \neq \beta_i$ for all i, i.e. α_1 is distinct from all other memory addresses. By proposition 4, neither $P_2 \rightarrow (I(\alpha_1) = I(\beta_i))$ nor $P_2 \rightarrow (I(\alpha_1) \neq I(\beta_i))$ is valid. Hence, the case where $I(\alpha_1)$ is distinct from all other addresses in memory in $P(s_2, g)$ is possible. Create a new address and data value in each state, and extend I by mapping the new addresses and data values to each other respectively. By repeated application of proposition 5, $R(u_1, u_2)$ holds.

6. g is a write. Similar to the case of read (QED).

In the following propositions, let P_1, P_2, Q be sets of predicates, r and q be individual predicates.

Proposition 2 If $P_1 \equiv P_2 \cup q$ and $P_1 \rightarrow r$ is valid, then $P_2 \rightarrow \bar{r}$ is not valid.

Proof Assume $P_2 \rightarrow \bar{r}$ is valid. This implies $P_2 \wedge q \rightarrow \bar{r}$ is valid, which in turn implies $P_1 \rightarrow \bar{r}$ is valid. This is in contradiction to $P_1 \rightarrow r$ being valid (QED).

Proposition 3 Let $P_1 \equiv P_2 \cup q$, and assume $P_1 \rightarrow r$ is valid, but neither $P_2 \rightarrow r$ nor $P_2 \rightarrow \bar{r}$ is valid. Then, $P_1 \equiv P_2 \cup q \cup r$.

Proof To show $P_1 \rightarrow P_2 \wedge q \wedge r$ is valid, assume $I(P_1) = T$ for some interpretation I. We need to show $I(P_2) = T$, $I(q) = T$, and $I(r) = T$. $P_1 \equiv P_2 \cup q$ and $I(P_1) = T$ implies $I(P_2) = T$ and $I(q) = T$. $I(P_1) = T$ and $P_1 \rightarrow r$ being valid implies $I(r) = T$. To show $P_2 \wedge q \wedge r \rightarrow P_1$ is valid, assume $I(P_2) = T$, $I(q) = T$, and $I(r) = T$ for some interpretation I. We have to show $I(P_1) = T$, which follows from $P_1 \equiv P_2 \cup q$, $I(P_2) = T$, and $I(q) = T$ (QED).

Proposition 4 If $P_1 \equiv P_2 \cup q$ and $P_1 \rightarrow r$ is not valid, then $P_2 \rightarrow r$ is not valid either.

Proof Let interpretation I be such that $I(P_1) = T$ and $I(r) = F$. By $P_1 \equiv P_2 \cup q$, $I(P_2) = T$. $I(P_2) = T$ and $I(r) = F$ implies $P_2 \rightarrow r$ is not valid (QED).

Proposition 5 $P_1 \equiv P_2 \cup q$ implies $P_1 \cup r \equiv P_2 \cup q \cup r$.

Proof Follows from the definition of equivalence of sets of predicates (QED).

Theorem 6 A state s is trace equivalent to a state t obtained from s by deleting an inaccessible location (a, γ).

Proof We will show $F(s) \subseteq F(t)$; the reverse direction is similar. Let $R(s_1, s_2)$ hold iff s_2 is the result of deleting an inaccessible location from s_1. Let $R(s_1, s_2)$ and $T(s_1, a, t_1)$ hold, where a is an assignment of values to the finite variables. It suffices to show that there exists t_2 such that $T(s_2, a, t_2)$ holds, and either $F(t_1) \subseteq F(t_2)$, or $R(t_1, t_2)$ holds. Proceeding by induction, assume that before processing a gate g, the partial state $P(s_2, g)$ is the result of deleting an inaccessible location from $P(s_1, g)$.

We define the following notations for the rest of the proof. Let P_1 and P_2 be the predicates in $P(s_1, g)$ and $P(s_2, g)$

respectively. Let I be an isomorphism which maps $P(s_1, g)$ after (a, γ) has been deleted to $P(t_1, g)$. Let $l_a(s)$ be the mapping which first deletes (a, γ) from s, and then applies I to the result. Let the two new partial states after processing g be u_1 and u_2. Based on g, we have the following cases:

1. g is a finite gate. Since by the inductive assumption the inputs to g are the same, assign the same value to the outputs.
2. g is a constant creator. Have the outputs of g map to one another, as an extension of the current isomorphism I.
3. g is an integer function. Just apply the function in both cases.
4. g is an integer predicate. By th9e inductive assumption and by lemma 3.1, we have that the output of g is determined by P_1 iff the output of g is determined by P_2, and they are the same. If both true and false cases have to be considered, have the corresponding cases map to one another, as far as bisimulation is concerned.
5. g is a read. Let the address being read in $P(s_1, g)$ be α_1. We have the following cases.

5a. $P_1 \rightarrow (\alpha_1 = \beta_1)$ is valid for some address β_1. Since a does not appear in any of the predicates, β_1 is not a. Hence $I(\beta_1)$ is well-defined, and $P_2 \rightarrow (I(\alpha_1) = I(\beta_1))$ is valid. By the inductive assumption, the data value at α_1 is mapped by I to the data value at $I(\alpha_1)$.

5b. $P_1 \rightarrow (\alpha_1 \neq \beta_i)$ is valid for all addresses β_i. This situation is not possible with the reason being as follows. Since a is one of the β_i's, and since a does not appear in any of the predicates of P_1, $P_1 \rightarrow (\alpha_1 \neq a)$ is valid iff α_1 is equal to one of the β_i's, which is a contradiction to the assumption that α_1 is distinct from all addresses β_i.

5c. For $\gamma_i \in \{\gamma_1, \ldots, \gamma_k\}$, neither $P_1 \rightarrow (\alpha_1 = \gamma_i)$ nor $P_1 \rightarrow (\alpha_1 \neq \gamma_i)$ holds, but we have assumed $\alpha_1 = \gamma_j$ for some γ_j. Hence, $P_1 \wedge (\alpha_1 = \gamma_j) \wedge \prod_{i \neq j} (\alpha_1 \neq \gamma_i)$ is satisfiable.

5c.1. γ_i is not a. Since P_1 and P_2 are isomorphic, $P_2 \wedge (I(\alpha_1) = I(\gamma_j)) \wedge \prod_{i \neq j} (I(\alpha_1) \neq I(\gamma_i))$ is also satisfiable, which implies we can assume $I(\alpha_1) = I(\gamma_j)$ holds in $P(s_2, g)$. By the inductive assumption, the data value at α_1 is mapped by I to the data value at $I(\alpha_1)$.

5c.2. γ_i is a. In this case, $I(\alpha_1)$ being distinct is possible in $P(t_1, g)$. Let the new location be $(\tilde{a}, \tilde{d})$, and the state obtained u_2. Consider the state v_2 obtained by adding the predicate $\tilde{d} = I(\gamma)$ to u_2. By lemma 3.6, $F(v_2) \subseteq F(u_2)$. By proposition 6, v_2 is trace equivalent to a state w_2 in which $\tilde{d}$ has been replaced by $I(\gamma)$. The predicate $\tilde{d} = I(\gamma)$ is dangling in w_2 and can be deleted by theorem 5. Let the resulting state be called y_2. We have that u_1 and y_2 are isomorphic (extend I by mapping a to $\tilde{a}$), which by theorem 2 implies $F(u_1) = F(y_2)$. We conclude $F(u_1) \subseteq F(u_2)$, as was desired.

5d. For $\gamma_i \in \{\gamma_1, \ldots, \gamma_k\}$, neither $P_1 \rightarrow (\alpha_1 = \gamma_i)$ nor $P_1 \rightarrow (\alpha_1 \neq \gamma_i)$ holds, but we have assumed α_1 is distinct from all other addresses. Hence, $P_1 \wedge \prod_{i=1}^{k} (\alpha_1 \neq \gamma_i)$ is satisfiable. Without loss of generality, let $\gamma_k = a$. We have $P_1 \wedge \prod_{i=1}^{k} (I(\alpha_1) \neq I(\gamma_i))$ is also satisfiable, which implies we can assume $I(\alpha_1)$ is distinct in $P(s_2, g)$ as well. In both cases a new location and data value is created. Extend I by mapping these locations and data values to each other respectively.

6. g is a write. Similar to the case of read (QED).

Proposition 6 Let a be a constant and γ an ICS term. If $a \doteq \gamma$ is a predicate in s, then replacing all occurrences of a with γ results in a trace equivalent state.

Proof Similar to the proof of theorem 3 (QED).