Interfacing System Description Languages to State-Space Exploration-Based Verification

Wendell Baker
September 2, 1993

This work identifies three techniques that can be used to improve query processing in a design verification system based on language-containment. The techniques allow synchronous systems with large structured state-spaces to be handled more predictably in the reachable-states computation and language-containment check [KH90][TBK91]. The first technique is a restriction on the synchronous system description along two axes: the patterns of inter-process communication and the variety of intra-process transition relations allowed. This restriction is embodied in the *Non-Deterministic Abstract Machine (NDAM)* architecture and notation. The second two techniques are *cofactor tuples* and *dirty BDD operations* both of which take advantage of the structure imposed by first restriction and allow for larger system descriptions to be processed faster in the reachable-states computation.

Background

Imperative and process-oriented system description languages such as VHDL, Verilog or Esterel[BC84] can be used for the synthesis of hardware, or embedded software as applicable, under the restriction of synchronous execution and finite state. Systems described in these languages are intrinsically characterized in terms of their interactions with their environment, the interactions among the processes of the system as well as the computations performed in the processes of the system.

In addition to the use of these languages for synthesis, it is attractive to be able to accept programs written in these languages as the direct input to a design verification system. In such a design verification system, properties about system models are proved through an exhaustive exploration of the reachable state-space of the model. Queries may involve safety properties, or assertions, which are merely questions about states reached during system execution. Other queries may involve liveness properties or eventualities which are questions about the behavior of the system as time goes to infinity. A simple reachable-states computation allows for the former to be answered by checking whether the states described by an assertion do not intersect with the states reachable by the system under any input. The latter sort of properties, being phrased in terms of the infinite behavior of the system, require more complex procedures such as language-containment [K90] which search for cycles of states that exist in the system but not in the query property.

Both the language-containment check and simple assertion-checking require the computation of the set of reachable states of the system. The efficient treatment of very large state spaces is accomplished through the use an implicit or symbolic representation of the system state. The state set and the state-transition relation is encoded by its characteristic function in a Binary Decision Diagram (BDD)[Bry89] and all state-manipulation operations are performed symbolically in this representation. The reached-states and language-containment check then depend upon the two

following two equations respectively:

$$R^*(curr) = \mu R, R_0(curr) \vee (\exists curr, R(curr) \wedge T_{SYS}(curr, next)) [next/curr]$$

$$T^*(curr, next) = \mu T(T_{SYS}(curr, next) \vee \exists int(T(curr, int) \wedge T(int, next)))$$

The first equation describes computation of reachable states of the system, R^*, as the least fixed point of a function that identifies the start states, R_0, or a way to get from the current-state set, R, to the next-state set by one step through the transition relation, T_{SYS}. All of the state formulae R^*, R_0, and R are expressed in terms of the current-state variables $curr$; the transition relation is expressed in terms of the current-state variables, $curr$, and the next-state variables, $next$. The second equation describes the computation of the transitive closure of the transition relation, T_{SYS}, as the fixed-point of a function that identifies transitions consisting of single-step transition using the basic system transition relation, T_{SYS}, directly or by using the intermediate relation, T, twice with using an intermediate state, int.

The importance of these two formulae is that they express conditions used in the assertion checking and language containment checks. Additionally the operations of *or*, *and*, *smooth* ($\exists x F(x, y)$) and *fixed-point* ($\mu F, I(x) \vee F(x)$) can all be expressed implicitly using BDDs [CBM89][M92] which allows for the efficient implementation of computation of these algebraic identities. The first equation is used directly in safety property verification by checking whether the states of an assertion intersect the reachable states of the system: $R_{\text{assert}}(curr) \cap R^*(curr) \neq \emptyset \Rightarrow \text{FAIL}$. The second equation is used in the language containment check as a basis for a description of the cycles of behavior that can occur in the system. The formula $\exists next, T^*(curr, next) \wedge T^*(next, curr)$ expresses the cycles that can occur in the system and is used in the language containment check to identify cycles of behavior in the system that are not contained in any query [TBK91].

While implicit techniques are very powerful, they suffer from instability because they are very sensitive to the ordering of the variables chosen for the BDD representation. A decent ordering must be determined for both the BDD variable sets $curr$, $next$ and int as well as the computation of the intermediate product machines that compose T_{SYS}. A bad ordering can lead to an exponential explosion in the BDD representing R^*, T_{SYS} or T^*. Because past approaches have used a direct interpretation of the system description in terms of a synchronous digital circuit, the verification algorithms for those approaches must be accompanied by heuristics to mitigate the ordering instability problems. The goal of this work is to investigate a formulation of the design verification problem which does not suffer from these instabilities.

This work focuses on the thesis that a verification query can be solved more efficiently by mapping the system description from its original form into one that is known to have good properties for the verification computations. Previous verification techniques on the other hand have used a representation of verification problem in terms of the synchronous circuit *directly* implied by the system description. This is a circuit which can have arbitrary structure and communication properties. Here, the idea is to simulate the provided system description on a synchronous architecture that is known to have good properties for the reachable-states and transition relation transitive-closure computations as implemented symbolically using BDDs.

Contributions

The central theme of this work is that design verification queries for arbitrary synchronous systems may be formulated for more efficient solution in terms of communications and operations on a known system architecture rather than in terms of a direct analogy with a digital circuit. In particular the intent is to be able to provide faster results for larger systems by mapping an arbitrarily-structured synchronous system description onto a larger but better-structured synchronous architecture. That architecture is better-structured in the sense of being known to have good properties for the two main computations of concern in language-containment based design verification: the reachable states computation and the transition relation transitive closure computation. The goal of this mapping is to offer a more stable base upon which design verification queries can be formulated.

1. Non-Deterministic Abstract Machine (NDAM)

The virtual system architecture proposed has much the same form as that of a clock-cycle based simulator or a synchronous software program. Examples of such architectures in real-world systems include programs written in a the Synchronous VHDL subset [Baker92] or any Esterel program [BC84]. The virtual architecture consists of a number of *processes* each of which has its own local memory that can only be manipulated by it. Processes communicate with each other and with the system environment by means of valued *signals* which are available to all other processes as a broadcast medium. Signals are statically allocated resources and maintain a notion of *event occurrence.* Time is discrete and is marked off by *instants* between which nothing else of interest occurs in the system. For any signal an event may or may not have occurred in the current instant and a process may await events on one or more signals.

In order to be mapped onto the NDAM architecture any system description is required to have a *causal ordering* among its process components. This is a mild restriction on communication paths that ensures there are no indeterminacies (race conditions) that depend upon process execution order. Causality ensures that there is at least one consistent ordering and there are no inconsistent orderings to the process components. The existence of a causal ordering allows the declared parallel execution of process components to be serialized. Serialization in turn allows the technique of early quantification to be used to simplify the transition relation transitive-closure computation of the language-containment check. Further, inside a process component the variety of transition relations allowed is restricted to a fixed set; this has the effect of defining an *instruction set* in which the system's processes must be expressed. These two restrictions in concert enforce a structure on the system description where an optimal ordering for BDD corresponding to the reachable-state set and system transition relation can be identified.

Cofactor Tuples

Cofactor tuples are used to speed up the next-state computation for well-characterized transition relations such as those corresponding to certain operations in an instruction set processor. The cofactor tuples are a method for hardcoding these transition relations using tuples of integers to represent the nodes of a transition relation BDD. This representation is possible because these transition relations are only used in the *and_smooth* portion of the next-state computation and in that context, only the *cofactor* operation is needed on the representation. The integer tuple representation is significant for it allows very large but highly-regular transition relations to be repre-

sented compactly outside the BDD system yet to operate consistently with a BDD-based symbolic *and_smooth* computation. The savings of this representation can be quite significant in both space and time. For example, the size of a BDD representation for a single-register writeback to a register file using the best possible ordering is $O(|REGISTERS|^2 \times |BITS|)$. Quite a number of the transition relations in the NDAM architecture have the necessary highly-regular properties.

Dirty BDD Operations

Dirty BDD operations are an *ad-hoc* method (*i.e.* they have no strong mathematical basis) for implementing the next-state computation on highly-structured memory-based systems. The NDAM architecture is memory-based in this sense as it contains a global memory for the per-process variables and a register file and operations unit which provide for instruction execution. The utility of dirty operations in the NDAM is to provide an efficient technique for implementing the next-state computation corresponding to the *readfrom* and *writeback* between the register file and its per-process memory plane. While the description of the readfrom and writeback operations as a transition relation in the BDD system is possible the size of the BDD makes this prohibitive. For example, for the case of BDD representation of a writeback operation where a whole register file is written to a memory consisting of independent planes, the BDD size using the best-case ordering is proportional to $O(|PLANES|^2 \times |REGISTERS| \times |BITS|)$. Dirty operations provide savings in this case by allowing this specific transition relation to be implemented as direct pointer manipulations on the BDD representation of the memory states.

Dirty operations are so named because they use direct, destructive pointer assignments inside the BDD Heap to emulate, under certain highly-constrained conditions, the results of the *and*, *smooth* and *substitute* operations of next-state computation. They allow the two operations of interest on memory structures, *readfrom* and *writeback* to be performed extremely efficiently. They are efficient in the sense that they only require a single node of new or intermediate storage and operate in time proportional only to the size of the memory plane being operated upon. The new BDD space required is therefore constant and trivial and the time required is independent of the size of the memory BDD and of the size of the readfrom or writeback transition relation BDD. Additionally the transition relation BDD of the operation being emulated need never be materialized.

References

[Baker92] Wendell Baker, *An Application of the Synchronous/Reactive Semantics to the VHDL Language*, Masters Report, University of California, Berkeley, December 1992.

[BC84] Gérard Berry and Laurent Cosserat. “The ESTEREL Synchronous Programming Language and its Mathematical Semantics.” In *Seminar on Concurrency*, edited by S.D. Brookes, A.W. Roscoe, and G. Winskel, pages 389-448. Springer-Verlag, 1984.

Bry89 R.E. Bryant, Graph Based Algorithms for Boolean Function Manipulation. In I*EEE Transactions on Computers*, Vol. C-35, No. 8, pages 677-691, August 1986.

[CBM89] O. Coudert, C. Berthet and J.C. Madre, Verification of SEquential Machines Based on Symbolic Execution. In *Proceedings of the Workshop on Automatic Verification*

Methods for Finite State Systems, Grenoble France, 1989

[K90] Robert Kurshan, Analysis of Discrete Event Coordination, In *Lecture Notes in Computer Science #430*, pages 414-453. Springer-Verlag, 1990.

[KH90] Robert Kurshan, Zvi Har'El, Software for Analytical Development of Communication Protocols, *AT&T Technical Journal,* pages 45-59, January 1990.

[M92] Kenneth McMillan, *Symbolic Model Checking*, Ph.D. Thesis, Carnegie Mellon University, 1992

[TBK91] Hervé Touati, Robert Brayton, Robert Kurshan, Testing Language Containment for ω-Automata using BDDs, In *International Workshop on Formal Methods in VLSI Design*, 1991.