Decomposing Image Computation for Symbolic Reachability Analysis Using Control Flow Information[*]

David Ward
IBM Printing Systems Division and
University of Colorado at Boulder

Fabio Somenzi
University of Colorado at Boulder

Abstract

The main challenge in BDD-based symbolic reachability analysis is represented by the sizes of the intermediate decision diagrams obtained during image computations. Methods proposed to mitigate this problem fall broadly into two categories: Search strategies that depart from breadth-first search, and efficient techniques for image computation. In this paper we present an algorithm that belongs to the latter category. It exploits define-use information along executable paths extracted from the control-flow graph of the model being analyzed; this information enables an effective constraining of the transition relation and a decomposition of the image computation process that often leads to much smaller intermediate BDDs. Our experiments confirm that this reduction in the size of the representation of state sets translates in significant decreases in CPU and memory requirements.

1. Introduction

Symbolic algorithms for reachability analysis have played a significant part in increasing the ability of model checkers to deal with large designs. Two main flavors of symbolic algorithms are currently in use for finite-state systems: BDD-based [17, 7] and SAT-based [1], each with its distinctive strengths and weaknesses. In particular, BDD-based algorithms compute and store the characteristic functions of sets of states. These characteristic functions are represented by Binary Decision Diagrams [3], which are canonical representations of Boolean functions.

One premise behind the symbolic approach is that the representations of the characteristic functions have sizes that are not strongly correlated to the cardinalities of the sets they represent. In particular, very large sets may have very concise descriptions. There is however, a flip side to that coin: Even if the set of all states reachable from the initial states of a model may have a very compact representation, subsets that are manipulated during reachability analysis may not enjoy that property and may cause a model checker to run out of memory or spend inordinate amounts of time.

The problem is further compounded by the fact that intermediate results may be sets of transitions rather than sets of states. The BDDs for them are often an order of magnitude larger than those for the state sets. Specifically, symbolic reachability analysis usually consists of a sequence of steps called *image computations*. If each step operates on the new states reached at the previous step, the result is a symbolic breadth-first search (BFS) of the state space. BFS is indeed the most common search strategy with BDDs. Each step determines the successors of a set of states according to a given transition relation. This transition relation is often represented as the implicit conjunction of a set of BDDs [31, 4], and the computation of successors proceeds by a sequence of conjunctions and quantifications. The largest BDDs manipulated by a symbolic reachability analysis algorithm are often the results obtained halfway through this sequence of operations.

Considerable attention has been devoted to this problem, and solutions have been proposed along two main lines of attack.[1] On the one hand, one can introduce flexibility in the search strategy by abandoning a strict BFS strategy. Instead, the states to be explored are chosen so as to keep their representation manageable [26, 5, 27, 10, 34]. The other approach, which this paper, in particular, is concerned with, applies disjunctive decomposition to image computation [6, 23, 22, 12, 18, 19, 9].

While the overall search strategy may still be BFS, the focus is on reducing the size of the intermediate BDDs of image computation. Partitioning affects the sizes of all BDDs involved in the computation: the set of states whose image is computed, the transition relation, and, often more pronouncedly, the intermediate results. The large impact on intermediate BDDs can be understood by considering that the commonly used conjunctive partitioning applied to the transition relation is often effective in containing the size of its representation. However, if a good quantification schedule cannot be found [11], the explosion in BDD size that has been prevented by conjunctive partitioning in the transition relation is only postponed to the image computation stage.

The methods proposed for disjunctive partitioning can be classified according to the criteria that guide the decomposition of the computation. One approach is to identify state variables that lead to an orthonormal decomposition of the characteristic functions such that the sizes of the components are heuristically minimized [6, 23, 25, 12]. Another approach looks at the dependencies in the transition relation and partitions either with the intent of breaking such dependencies [22] or to restrict them inside the component blocks [18, 20].

This paper presents a new approach to disjunctive partitioning and image computation that relies on program analysis, and in particular on control flow information extracted from the Register Transfer Level (RTL) description of a hardware model.[2] By analyzing the RTL description we leverage the natural seapration of control and data present in the source code. As observed in [34], paths in the Control Flow Graph (CFG) of a model help one identify the

[*]This work was supported in part by SRC contract 2006-TJ-1365.

[1]Approaches based on functional dependencies [16, 32] are also effective in some cases.

[2]Though the general ideas discussed in this paper can be applied to software as well as hardware model checking, the discussion is here limited to the latter. Specifically, we use Verilog [30] as our input language.

modes of operation of a model. The enabling conditions of these paths can then be used to constrain the transition relation of a model so as to achieve a disjunctive partitioning. The simplifications that occur in the transition relation as a result of the restriction to one mode of operation are often conducive to great simplification of the BDDs because they naturally tend to reduce dependencies among variables [27]. It is therefore often possible to achieve dramatic decreases in time and memory resources required for reachability analysis.

As an extreme case, consider a model with two n-bit registers. At each execution step, the contents of one register are copied to the other register while being rotated by an amount specified by a primary input. The contents of the first register are also updated with the value of a primary input. Because of the rotation, each bit of the second register depends on every bit of the first register. The transition relation cannot be represented as a single BDD even for small values of n and, what is more important, image computation according to the standard algorithm quickly runs out of memory as n increases. However, if the image computation is decomposed according to the rotation amount, then in each part every bit of the second register depends on exactly one bit of the first. This greatly simplifies the computation.

A key piece of information derived from the analysis of the control flow of the program concerns the definitions and uses of variables. It is not uncommon for most variables to be neither used nor defined along a given path. When the transition relation is restricted to that path, the variables that are not defined are known to retain their values. Our image computation algorithm exploits this knowledge to reduce the intermediate BDDs and alleviate the cost of variable substitutions.

Since the CFG is not a canonical representation of a model, the effectiveness of the approach we propose depends to some extent on how the model is described at the RTL. Its effectiveness also varies from model to model. Certain models are bound to remain intractable in spite of disjunctive partitioning in image computation because they are inherently unsuitable for breadth-first analysis. For yet other models, any attempt at exactly representing the set of reachable states as a BDD are doomed to failure. For instance, if the set of reachable states does not have a concise BDD representation under any variable order, changing the search strategy can only lead to more states being reached within the given computational resources. Notwithstanding these limitations, disjunctive partitioning based on control-flow information achieves remarkable results in many cases. Our preliminary experimental results show that it significantly extends the applicability of symbolic model checkers.

The use of control-flow information to speed up model checking was advocated in [9, 34]; the latter uses the control flow graph to guide symbolic guided search. The approach of [9] is based on disjunctive decomposition and breadth-first search, but no use is made of information about definitions and usages of variables to optimize image computation.

2. Preliminaries

2.1 Binary Decision Diagrams

Binary Decision Diagrams (BDDs [3]) are a graphical representation of Boolean functions. A BDD is derived from a binary decision tree by merging isomorphic subgraphs and eliminating redundant nodes. For a given variable order, this derivation results in a canonical representation. As a consequence, equivalence tests are efficient, and, thanks to recourse to memoization, the algorithms that operate on BDDs are fast. Large sets can be manipulated via their characteristic functions, which in turn can be represented by BDDs. BDDs are used to represent sets of states and transitions in symbolic model checking.

2.2 Symbolic Reachability Analysis

We consider symbolic reachability analysis of transition systems with finite sets of states Q and inputs W defined by a *transition relation* $T \subseteq Q \times W \times Q$ and *initial state set* $I \subseteq Q$. A triple (q_1, w, q_2) is in T if and only if the transition system can proceed from q_1 to q_2 when the input is w; in this case q_2 is a *successor* of q_1. State q is *reachable* from state q' if there exists a sequence of states $q_1, \ldots, q_n$ such that $q = q_1$, $q' = q_n$, and for $1 < i \leq n$, q_{i+1} is a successor of q_i. The reachability analysis problem consists of finding all states that are reachable from some state in I. For $S \subseteq Q$, let $\mathsf{EY}\, S$ denote all states that are successors of some state in S. Then reachability can be computed as a fixpoint:

$$\mu Z \,.\, I \cup \mathsf{EY}\, Z \ \ .$$

Let $Z_0 = I$ and, for $i \geq 0$, $Z_{i+1} = I \cup \mathsf{EY}\, Z_i$ be the iterates of the computation. Then Z_i contains the states that can be reached in at most i steps from states in I. Hence, the fixpoint computation corresponds to *breadth-first search* (BFS) of the transition system starting from the initial states. In symbolic model checking, transition relations and sets of states are represented by their characteristic functions, which can be manipulated in various forms. In this paper we assume that (reduced, ordered) BDDs are used for this purpose. Success with symbolic computations depends on the algorithm's ability to keep the BDDs small. Several factors affect the size of BDDs, including the variable orders. In this paper, however, we focus on the impact on the BDD sizes of the procedure employed for the computation of $\mathsf{EY}\, Z$—the so-called *image computation*.

We assume that the transition relation T is complete, that is,

$$\forall q_1 \in Q \,.\, \forall w \in W \,.\, \exists q_2 \in Q \,.(q_1, w, q_2) \in T \ \ ,$$

and that it is represented by a predicate $T(x, w, y)$, where x is a vector of current state variables, w is a vector of primary inputs, and y is a vector of next state variables. All variables are binary. Without loss of generality, we assume a binary encoding for the states and the inputs, so that the x and y variables range over the encoding of the states and the w variables range over the encodings of the inputs. In what follows we do not distinguish states and inputs from their encodings. $T(x, w, y)$ is true when the valuations of x, w, and y correspond to a transition in T.

The transition relation predicate is often represented in a conjunctively decomposed form:

$$T(x, w, y) = \bigwedge_i T_i(y^i, w, x) \ \ ,$$

where each T_i defines a subvector y^i of the next state variables such that $\{y^i\}$ is a partition of y. (More general definitions of conjunctive partitioning are possible, but will not be considered in this paper.)

Image computation with a conjunctively partitioned transition relation consists of evaluating

$$Z_{i+1}(y) = \exists x, w \,.\, \bigwedge_i T_i(y^i, w, x) \wedge Z_i(x) \ \ ,$$

where quantification of variable vectors is shorthand for the quantification of all the variables in the vector,

$$\exists x \,.\, f = \exists x_1 \,.\, \exists x_2 \,.\, \cdots \,.\, f \ \ ,$$

and $Z_i(x)$ is the characteristic function of the states whose successors are sought. For the result $Z_{i+i}(y)$ to be employed in the next image computation, it must be converted to $Z_{i+i}(x)$ by variable substitution. It should be noted that this substitution, harmless as it may seem, can be problematic, especially with regard to BDD variable reordering [13]. Finally, it should be noted that variables are in practice quantified as soon as possible [11, 21], as this normally helps in keeping the BDDs sizes under control. A variable can be quantified as soon as it appears only in one conjunct. Current BDD packages in fact provide operations that quantify variables while conjoining two BDDs.[3]

Two elementary properties of images that will be useful in describing the algorithm are that the image of the empty set is the empty set, and that image computation distributes over disjunction, that is:

$$\exists x, w \,.\, T(y,w,x) \land (A(x) \lor B(x)) =$$
$$(\exists x, w \,.\, T(y,w,x) \land A(x)) \lor (\exists x, w \,.\, T(y,w,x) \land B(x)) \ .$$

Distributivity is the foundation for disjunctive partitioning.

2.3 Control Flow Graph (CFG)

Many program analysis techniques work on graphs derived from the program text. Among these, the CFG is a directed graph that represents the flow of control of a program (hardware or behavioral model). Each node represents an assignment or branching statement S_i in a program P. Each directed arc represents flow of control from one node to another. A CFG can be extracted in a single pass traversal over P. In our implementation we create one CFG for each Verilog *always block*. The set of CFG paths in a program P consisting of multiple *always blocks* is simply the Cartesian product of the set of paths computed from each *always block* CFG. In the rest of the paper we will refer to CFG paths as the set of paths in P computed by taking the Cartesian product of the paths of the program CFG(s)(which may contain a single or multiple *always block*).

Definition 1 (Control Flow Graph (CFG)) *A control flow graph CFG is a directed graph $G = (V, E)$, where: (1) V is a finite set of nodes including two distinguished nodes of type* entry *and* exit*. All other nodes are of one of two types:* assignment *and* decision*. (2) $E \subset V \times V$ is a control flow relation, whose elements are directed edges.*

We assume that the flow relation obeys restrictions. Specifically, we assume that the arcs can be partitioned into forward arcs (E_{fwd}) and back arcs (E_{bwd}) so that the forward arcs form a DAG in which all nodes are reachable from the entry node. We also assume that the exit node is reachable from all nodes in the CFG. Furthermore, each back edge goes from a node to another that dominates it. These assumptions imply *reducibility* of the CFG. Intuitively, the different types of nodes map to the basic types of statements, and in fact we shall call the CFG nodes *statements*. The edges in E represent the transfer of control between statements.

Definition 2 (Def/Use Graph) *A def/use graph G_{DU} is a structure $\langle CFG, \Sigma, D, U\rangle$ where Σ is a set of variables (registers) in a program P, $D : V_{CFG} \mapsto \delta(\Sigma)$ and $U : V_{CFG} \mapsto \theta(\Sigma)$ are functions mapping nodes V of CFG, i.e, V_{CFG} to the set of variables defined (δ) or used (θ) in the statements corresponding to the nodes V, respectively.*

[3]In all our experiments, we use early quantification, though for some models, like the two-register one discussed in the introduction, it is ineffective.

Table 1: Control Flow Analysis

CFG Path	PATH Predicate	Path Defines	Path Uses	Path Inerts
1	$\texttt{rst} \land \neg\texttt{state} \land \texttt{a} = 0$	a,b,c,state	—	—
2	$\texttt{rst} \land \neg\texttt{state} \land \texttt{a} \neq 0$	a,b,c,state	—	—
3	$\texttt{rst} \land \texttt{state} \land \texttt{c} < 4095$	a,b,c,state	—	—
4	$\texttt{rst} \land \texttt{state} \land \texttt{c} \geq 4095$	a,b,c,state	—	—
5	$\neg\texttt{rst} \land \neg\texttt{state} \land \texttt{a} = 0$	a,state	—	b,c
6	$\neg\texttt{rst} \land \neg\texttt{state} \land \texttt{a} \neq 0$	b,state	a	c
7	$\neg\texttt{rst} \land \texttt{state} \land \texttt{c} < 4095$	c,state	—	a,b
8	$\neg\texttt{rst} \land \texttt{state} \land \texttt{c} \geq 4095$	state	c	a,b

Definition 3 *A path π is a list of nodes $(v_1 \ldots v_k)$ such that v_1 is the* entry *node, and $\forall i, 1 \leq i \leq k-1, (v_i, v_{i+1}) \in E_{fwd}$. A path π represents one clock cycle of a Verilog always block.*

The naive approach to determining the paths in a program P would be to enumerate all possible paths in the programs CFG. This would result in the worst case in up to 2^n paths, where n is the number of decisions in the program text. However, we employ a BDD-based traversal algorithm to compute each path's enabling predicate. (Decision node predicates are represented by BDDs.) The path enabling predicate is simply the conjunction of the predicates along the path. This algorithm has the characteristic that as false paths are identified (a path whose path predicate is false), they are eliminated from subsequent iterations thus decreasing the number of paths that need to be checked for falsehood in subsequent steps. Data-flow information is not considered while determining falsehood, thus some infeasible paths could possibly be missed. Our algorithm performs better in practice, however, it is still exponential in the worst case. We compute the set of path enabling predicates while parsing the program text without having to build the program CFG(s).

3. Algorithm

Given a model in an RTL language (Verilog in our case), our algorithm uses the path information in the CFG of the RTL description to speed up image computation, and hence reachability analysis.

The information about each path of the CFG that is used includes the path enabling predicate, which is the conjunction of the guards (the conditional expressions controlling program flow) along the path itself, and the set of variables that are defined and used along the path. The path enabling predicates depend on both state variables and primary inputs. Since the transition relation is complete, the projection of their union over the state space covers the entire space. Therefore they can be used for disjunctive partitioning of the image computations. Figure 1 shows a simple Verilog example description, while the corresponding CFG is shown in Fig. 2. The CFG in Fig. 2 shows only the edges in E_{fwd}. We omit the back edges E_{bwd} for clarity.

For the same example, Table 1 lists the paths of the CFG and for each of them reports the *path enabling predicate*, and classifies the state variables as *defines*, *uses*, and *inerts*. The table refers to variables that are bit vectors to avoid clutter; the translation to binary variables is straightforward.

For our purposes, a path in the CFG is an alternating sequence of nodes and arcs of the CFG that connects the entry node to the exit node. The path enabling predicate is the conjunction of all the conditionals that are true along the path. A variable of the CFG is *defined* along a path if it is the target of at least one assignment

```
module example(clock,rst,indat);
        input        clock;
        input[31:0]  indat;
        input        rst;

        reg          state;
        reg[31:0]    a,b,c;

        initial begin
          state=0;a=0; b=0; c=0;
        end

        always @ (posedge clock) begin
P1:       if (rst) begin
S1..S4:      a = 0; b = 0; c = 0; state=0;
          end
P2:       if (!state) begin
P3:          if (a == 0)
S5:            a=indat;
             else
S6:            b=a;
          end else begin
P4:          if (c < 4095)
S7:            c = c+1;
          end
S8:       state = ~state;
        end
endmodule
```

Figure 1: Example Verilog model

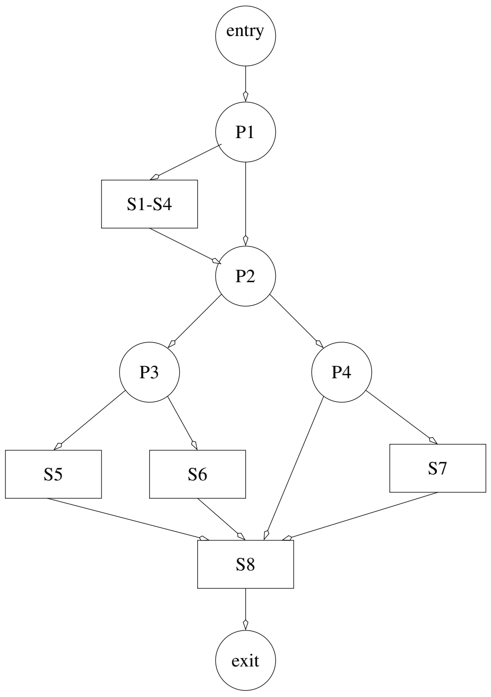

Figure 2: CFG for the Verilog model of Fig. 1

```
Reachability($S_0$, CFG, $T$)
$R = N = S_0$
while ($N \neq 0$) {
    $J = 0$
    foreach $\pi \in$ PATHS(CFG) {
        $N_\pi = N \wedge \exists$ inputs . PATH_COND($\pi$)
        if ($N_\pi \neq 0$) {
            $T_\pi$ = $T\wedge$ PATH_COND($\pi$)
            img = IMAGE($T_\pi$, $N_\pi$)
            $J = J\vee$ img
        }
    }
    $N = J \wedge \neg R$
    $R = R \vee N$
}
return $R$
```

Figure 3: Symbolic reachability algorithm with disjunctive image partitioning based on CFG paths

along the path. We denote the x variables defined along path π by $D_x(\pi)$. A variable is *used* along a path if it is not defined and it appears on the right-hand side of an assignment or in a conditional along the path. We denote the variables used along path π by $U_x(\pi)$. Variables that are neither defined nor used along a path are called *inert* for that path; the variables inert along path π are denoted by $I_x(\pi)$.

The pseudocode of the symbolic reachability algorithm is shown in Fig. 3. It is similar to the standard symbolic BFS algorithm, except for the fact that the image computations are decomposed according to the paths of the CFG. For each path, the enabling condition is conjoined with the transition relation to produce a constrained relation T_π.

The states whose image must be computed are also conjoined with the path enabling predicate. Oftentimes the conjunction is false, and the corresponding image computation is skipped.

The correctness of the algorithm of Fig. 3 follows from the fact that image computation distributes over disjunction and that the disjunction of all the enabling predicates is the tautologous predicate. Some standard optimizations of symbolic BFS are omitted for clarity from Fig. 3. For instance, the choice of the next state frontier from the interval $[N, R]$. These optimizations can be applied with no adverse effect on the correctness of the algorithm.

If the CFG has many paths, it is possible for the inner loop of Fig. 3 to execute too many times. In such a case it is difficult to recover the overhead due to the many invocations of the image computation; it is often more efficient to revert to non-decomposed computation. The pseudocode does not show this option, which is however included in the algorithm that we implemented.[4]

The increased efficiency of the algorithm of Fig. 3 over standard symbolic breadth-first analysis comes from two sources. On the one hand, the partitioning of the image computation task decreases dependencies in the transition relation and simplifies in general the BDDs. On the other hand, the classification of variables into defines, uses, and inerts affords a more efficient computation of each part of the image as we now describe.

[4]The threshold for reverting to non-decomposed computation is a user-specified parameter. In our experiments we chose a value of 7500 paths, which was more than enough for all our designs.

For path π, the computation of procedure IMAGE amounts to

$$\mathrm{img}(D_y(\pi), U_y(\pi), I_y(\pi)) = \exists D_x(\pi) \,.\, \exists U_x(\pi) \,.\, \exists I_x(\pi) \,.\, \exists w \,. \\ T_\pi(x, w, y) \wedge N_\pi(D_x(\pi), U_x(\pi), I_x(\pi)) \ , \quad (1)$$

followed by the substitution of the y variables by the corresponding x variables in the result. We now observe that

$$T_\pi(x, w, y) = T'_\pi(D_x(\pi), U_x(\pi), w, D_y(\pi)) \\ \wedge \bigwedge_{x_i \in U_x(\pi) \cup I_x(\pi)} (y_i \leftrightarrow x_i) \ ,$$

because the values of the variables not defined along π are not affected by its execution. Substituting into (1), and taking into account that

$$\exists a \,.\, f(a) \wedge (a \leftrightarrow b) = f(b) \ ,$$

we get

$$\mathrm{img}(D_y(\pi), U_y(\pi), I_y(\pi)) = \exists D_x(\pi) \,.\, \exists w \,. \\ T'_\pi(D_x(\pi), U_y(\pi), w, D_y(\pi)) \wedge N_\pi(D_x(\pi), U_y(\pi), I_y(\pi)) \ , \quad (2)$$

whence, by variable renaming,

$$\mathrm{img}(D_y(\pi), U_x(\pi), I_x(\pi)) = \exists D_x(\pi) \,.\, \exists w \,. \\ T'_\pi(D_x(\pi), U_x(\pi), w, D_y(\pi)) \wedge N_\pi(D_x(\pi), U_x(\pi), I_x(\pi)) \ . \quad (3)$$

The advantage of (3) over (1) is twofold. On the one hand, the terms of the transition relation corresponding to used and defined variables have been removed. This lowers the number of variables that can be involved in the intermediate results and greatly helps in the reduction of the sizes of the intermediate BDDs. On the other hand, the result of image computation must be eventually expressed in terms of current state variables. Hence, the next state variables must be substituted. In (3) substitution is required only for the defines of the path. The main advantage comes from the simplification of the transition relation, but both can be significant.

In many cases, a further optimization is advantageous. Let us define

$$N^0_\pi(U_x(\pi), I_x(\pi)) = \exists D_x(\pi) \,.\, N_\pi(D_x(\pi), U_x(\pi), I_x(\pi)) \\ N^1(D_x(\pi), U_x(\pi), I_x(\pi)) = N_\pi(D_x(\pi), U_x(\pi), I_x(\pi)) \\ \downarrow N^0_\pi(U_x(\pi), I_x(\pi)) \ ,$$

where '$\downarrow$' denotes the *constrain* operator [7]. Since the constrain operator is such that $f \wedge g = f \wedge (g \downarrow f)$, it is immediately verified that

$$N_\pi = N^0_\pi \wedge N^1_\pi \ .$$

Since N^0_π does not depend on any of the variables to be quantified in (3), we obtain

$$\mathrm{img}(D_y(\pi), U_x(\pi), I_x(\pi)) = N^0_\pi(U_x(\pi), I_x(\pi)) \wedge \exists D_x(\pi) \,. \\ \exists w \,.\, T'_\pi(D_x(\pi), U_x(\pi), w, D_y(\pi)) \\ \wedge N^1_\pi(D_x(\pi), U_x(\pi), I_x(\pi)) \ . \quad (4)$$

The transformation of (3) into (4) reduces the intermediate BDDs and is advantageous when the conjunctive decomposition of N_π into N^0_π and N^1_π can be obtained cheaply. This is usually the case when the defines along path π are a small fraction of the state variables. (Not only the quantification tends then to be inexpensive, but the BDD for N^1_π is often very small.)

Table 2: Experimental results for reachability analysis

Circuits	FFs	Reachable States	CFG Paths	Times in seconds	
				New BFS	BFS
blackjack	104	2.443e+07	25	226.86	220.2
Palu	37	2.205e+09	19	118.39	1133.7
CRC	32	4.295e+09	4	Mem. out	Mem. out
BPB	36	6.872e+10	289	51.0	100.0
Rotator	64	1.845e+19	32	4.37	Mem. out
Vsa	66	1.625e+14	27	445.0	1943.36
B04	66	5.650e+15	27	250.36	4678.99
FIFOs	142	—	20	Timeout	62.53
Spinner	65	3.659e+19	32	4.54	Mem. out
twoQ	66	1.300e+17	63	1179.01	2304.87

Table 3: Peak Live Nodes

Circuits	FFs	Reachable States	Size in Kbytes	
			New BFS	BFS
blackjack	104	2.443e+07	9365	8810
Palu	37	2.205e+09	25520	47517
CRC	32	4.295e+09	—	—
BPB	36	6.872e+10	2538	1884
Rotator	64	1.845e+19	18	—
Vsa	66	1.625e+14	15733	29348
B04	66	5.650e+15	14715	32287
FIFOs	142	—	—	3437
Spinner	65	3.659e+19	68	—
twoQ	66	1.300e+17	20889	27666

Constrain is not the only operator that may be applied to the computation of N^1_π. For instance, the *restrict* operator [8] is a popular alternative because it often produces smaller BDDs. However, when the second operand to restrict is obtained by quantifying variables from the first operand, restrict gives no advantage over constrain.[5] Other so-called *generalized cofactor* algorithms (e.g., those of [28, 14] or the *squeeze* algorithm available in the CUDD package [29]) may produce smaller BDDs than constrain, but tend to be slower.

4. Experimental Results

We implemented our algorithm in VIS 2.1 [2]. We show experimental results on certain benchmarks from the VIS Verilog suite [33] that are known to be hard for BFS reachability analysis. In addition, we show results for some small or medium sized circuits where BFS reachability analysis does very well. Our experiments were run on a 2.0 GHz Pentium M machine with 2 GB of RAM running Linux. Each experiment was allotted a 7200 s time limit and the data segment size was limited to 1 GB (e.g., ulimit -d 1000000). Our results are compared to the standard BFS reachability algorithm implemented in VIS. The extraction time to retrieve control flow information needed by our algorithm was negligible compared to the time to complete reachability analysis and thus will be omitted.

We used ten circuits in our experiments. Blackjack is a model

[5]Restrict differs from constrain in that it applies local quantification to its second operand. When the second operand is obtained from the first by quantification, local quantification is never triggered.

of a blackjack card game keeping track of the cards dealt to a player and a dealer. Palu is an elementary pipelined ALU with the ability to stall. The pipeline consists of an ALU and a register file. At each clock cycle the pipeline starts the execution of an instruction, which completes in three cycles. CRC computes a 32-bit cyclic redundancy code of a stream of bytes. BPB is a branch prediction buffer that predicts whether a branch should be taken depending on the correctness of the previous predictions. Rotator and Spinner consist of a barrel shifter sandwiched between registers. Vsa is a simple non-pipelined microprocessor that executes 12-bit instructions—ALU operations, loads, stores, conditional branches—in five stages: fetch, decode, execute, memory access, and write-back. It has four registers, with one always set to zero. B04 is a Verilog translation of the original b04 circuit from the ITC99 benchmark set [15]. It computes the minimum and maximum of a set of numbers. FIFOs is a model to check the equivalence of two different FIFO implementations: 1) a shift register FIFO and 2) a Ring buffer FIFO. Table 2 compares standard BFS reachability analysis in which the image computation algorithm is the one of [24] with our new BFS algorithm. Columns 1, 2, and 3 give the name of the circuit, number of flip-flops (state variables) and number of reachable states of the circuit. Column 4 gives the number of CFG paths (from start node to end node) generated from the circuit's control structure. Columns 5, and 6 compare run times for reachability analysis for new BFS and standard BFS, respectively. Table 3 shows the peak live nodes during reachability analysis for both the new and standard BFS. These numbers are indicative of the sizes of the BDDs encountered during image computation. The disjunctive decomposition has an inherent overhead in terms of BDD nodes because it instantiates two transition relations simultaneously. This, for smaller examples, may lead to more BDD nodes being allocated than in ordinary BFS. However, the many cases in which the latter runs out of memory demonstrate the efficiency of our technique.

5. Conclusions

In this paper we have presented a new approach to the disjunctive decomposition of image computations based on control flow information extracted from the Register Transfer Level Description of the model. Paths in the Control Flow graph define a partition of the state and input space that often leads to dramatic decreases in the sizes of the intermediate BDDs built during image computations. The results we have presented are for the case of breadth-first reachability analysis, but the approach based on control-flow information extends also to other search strategies.

It is interesting, in particular, to compare the approach to symbolic reachability proposed in this paper to the one of [34], in which program analysis techniques are used to derive hints for guided search. Both approaches use paths in the CFG to constrain the transition relation. In [34], unlike here, the search strategy is not breadth-first. In some cases, as for CRC in Sect. 4, this allows reachability to complete even though BFS fails. In other occasions, as for B04, guided search encounters difficulties when the complete transition relation is restored, because the image of a large BDD must be computed. This problem is avoided with breadth-first search. The exploitation of the information on defines and uses along the various paths is the main novelty of this paper. Though we have demonstrated its advantages in the more common and controlled context of breadth-first search, it can be easily extended to the guided search approach. Producing shortest witnesses to reachability of a given state is also easier in the case of BFS analysis.

Comparing our approach to other techniques based on BFS and disjunctive partitioning of the image computations, we observe that the most effective techniques devised so far are in our experience those that attempt to break dependencies in the transition relation. In that respect, use of the CFG information is quite effective, and allows, in addition, simplifications of the computation that are not possible without the knowledge of which variables are left unchanged by a constrained partition relation. Such information is easily extracted from the CFG.

It is possible in principle to cluster several paths of the CFG and partition the image computation according to the clusters. This is a potentially profitable approach that we have not yet investigated. We also plan to study the benefits of our technique in the context of preimage computation (computation of the predecessors of a set of states) which is required by popular model checking algorithms.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, pages 193–207, Amsterdam, The Netherlands, Mar. 1999. LNCS 1579.

[2] R. K. Brayton et al. VIS. In *Formal Methods in Computer Aided Design*, pages 248–256. Springer-Verlag, Berlin, Nov. 1996. LNCS 1166.

[3] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.

[4] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 403–407, San Francisco, CA, June 1991.

[5] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitionining and partial iterative squaring: An effective approach for symbolic traversal of large circuits. In *Proceedings of the Design Automation Conference*, pages 728–733, Anaheim, CA, June 1997.

[6] G. Cabodi, P. Camurati, and S. Quer. Improved reachability analysis of large finite state machines. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–360, Santa Clara, CA, Nov. 1996.

[7] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using Boolean functional vectors. In L. Claesen, editor, *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, Nov. 1989.

[8] O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 126–129, Nov. 1990.

[9] X. Feng, A. J. Hu, and J. Yang. Partitioned model checking from software specifications. In *Proceedings of the 10th Asia and South Pacific Design Automation Conference (ASP-DAC 2005)*, pages 583–587, 2005.

[10] R. Fraer, G. Kamhi, B. Ziv, M. Y. Vardi, and L. Fix. Prioritized traversal: Efficient reachability analysis for verification and falsification. In E. A. Emerson and A. P. Sistla, editors, *Twelfth Conference on Computer Aided Verification (CAV'00)*, pages 389–402. Springer-Verlag, Berlin, July 2000. LNCS 1855.

[11] D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 299–310, Berlin, 1994. Springer-Verlag. LNCS 818.

[12] A. Gupta, Z. Yang, P. Ashar, and A. Gupta. SAT-based image computation with application in reachability analysis. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 354–271. Springer-Verlag, Nov. 2000. LNCS 1954.

[13] H. Higuchi and F. Somenzi. Lazy group sifting for efficient symbolic state traversal of FSMs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 45–49, San Jose, CA, Nov. 1999.

[14] Y. Hong, P. A. Beerel, J. R. Burch, and K. L. McMillan. Safe BDD minimization using don't cares. In *Proceedings of the Design Automation Conference*, pages 208–213, Anaheim, CA, June 1997.

[15] ITC'99 benchmark home page. http://www.cerc.utexas.edu/itc99-benchmarks/bench.html.

[16] J. C. Madre. Private communication, 1996.

[17] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[18] C. Meinel and C. Stangier. Speeding up image computation by using RTL information. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 443–454. Springer-Verlag, Berlin, Nov. 2000. LNCS 1954.

[19] C. Meinel and C. Stangier. Hierarchical image computation with dynamic conjunction scheduling. In *Proceedings of the International Conference on Computer Design*, Austin, TX, Sept. 2001.

[20] C. Meinel and C. Stangier. Hierarchical image computation with dynamic conjunction scheduling. Presented at IWLS01, June 2001.

[21] I.-H. Moon, G. D. Hachtel, and F. Somenzi. Border-block triangular form and conjunction schedule in image computation. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 73–90. Springer-Verlag, Nov. 2000. LNCS 1954.

[22] I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of the Design Automation Conference*, pages 23–28, Los Angeles, CA, June 2000.

[23] A. Narayan, J. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli. Partition ROBDDs: A compact, canonical and efficiently manipulable representation for boolean functions. In *Proceedings of the International Conference on Computer-Aided Design*, pages 547–554, Santa Clara, CA, Nov. 1996.

[24] R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. Presented at IWLS95, Lake Tahoe, CA, May 1995.

[25] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and decomposition of decision diagrams. In *Proceedings of the Design Automation Conference*, pages 445–450, San Francisco, CA, June 1998.

[26] K. Ravi and F. Somenzi. High-density reachability analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 154–158, San Jose, CA, Nov. 1995.

[27] K. Ravi and F. Somenzi. Hints to accelerate symbolic traversal. In *Correct Hardware Design and Verification Methods (CHARME'99)*, pages 250–264, Berlin, Sept. 1999. Springer-Verlag. LNCS 1703.

[28] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton. Heuristic minimization of BDDs using don't cares. In *Proceedings of the Design Automation Conference*, pages 225–231, San Diego, CA, June 1994.

[29] F. Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado at Boulder, ftp://vlsi.colorado.edu/pub/.

[30] D. E. Thomas and P. R. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, Boston, MA, third edition, 1996.

[31] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDD's. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 130–133, Nov. 1990.

[32] C. A. J. van Eijk. *Formal Methods for the Verification of Digital Circuits*. PhD thesis, Eindhoven University of Technology, Department of Electrical Engineering, Aug. 1997.

[33] Vis verification benchmarks. http://vlsi.colorado.edu/∼vis.

[34] D. Ward and F. Somenzi. Automatic generation of hints for symbolic traversal. In *Correct Hardware Design and Verification Methods (CHARME'05)*, pages 207–221, Saarbrucken, Germany, Oct. 2005. Springer-Verlag. LNCS 3725.