Optimizing Symbolic Model Checking for Constraint-Rich Models

Bwolen Yang Reid Simmons Randal E. Bryant
David R. O'Hallaron

March 1999

CMU-CS-99-118

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A condensed version of this technical report will appear in Proceedings of the International Conference on Computer-Aided Verification, Trento, Italy (July 1999).

Effort sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287, in part by the National Science Foundation under Grant CMS-9318163, and in part by grants from the Intel Corporation and NASA Ames Research Center. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

Keywords: symbolic model checking, Binary Decision Diagram (BDD), time-invariant constraints, redundant state-variable elimination, macro.

Abstract

This paper presents optimizations for verifying systems with complex time-invariant constraints. These constraints arise naturally from modeling physical systems, e.g., in establishing the relationship between different components in a system. To verify constraint-rich systems, we propose two new optimizations. The first optimization is a simple, yet powerful, extension of the conjunctive-partitioning algorithm. The second is a collection of BDD-based macro-extraction and macro-expansion algorithms to remove state variables. We show that these two optimizations are essential in verifying constraint-rich problems; in particular, this work has enabled the verification of fault diagnosis models of the Nomad robot (an Antarctic meteorite explorer) and of the NASA Deep Space One spacecraft.

1 Introduction

This paper presents techniques for using symbolic model checking to automatically verify a class of real-world applications that have many time-invariant constraints. An example of constraint-rich systems is the symbolic models developed by NASA for on-line fault diagnosis [16]. These models describe the operation of components in complex electro-mechanical systems, such as autonomous spacecraft or robot explorers. The models consist of interconnected components (e.g., thrusters, sensors, motors, computers, and valves) and describe how the *mode* of each component changes over time. Based on these models, the Livingstone diagnostic engine [16] monitors sensor values and detects, diagnoses, and tries to recover from inconsistencies between the observed sensor values and the predicted modes of the components. The relationships between the modes and sensor values are encoded using symbolic constraints. Constraints between state variables are also used to encode interconnections between components. We have developed an automatic translator from such fault models to SMV (Symbolic Model Verifier) [11], where mode transitions are encoded as transition relations and state-variable constraints are translated into sets of time-invariant constraints.

To verify constraint-rich systems, we introduce two new optimizations. The first optimization is a simple extension of the conjunctive-partitioning algorithm. The other is a collection of BDD-based macro-extraction and macro-expansion algorithms to remove redundant state variables. We show that these two optimizations are essential in verifying constraint-rich problems. In particular, these optimizations have enabled the verification of fault diagnosis models for the Nomad robot (an Antarctic meteorite explorer) [1] and the NASA Deep Space One (DS1) spacecraft [2]. These models can be quite large, with up to 1200 state bits.

The rest of this paper is organized as follows. We first briefly describe symbolic model checking and how time-invariant constraints arise naturally from modeling (Section 2). We then present our new optimizations: an extension to conjunctive partitioning (Section 3), and BDD-based algorithms for eliminating redundant state variables (Section 4). We then show the results of a performance evaluation on the effects of each optimization (Section 5). Finally, we present a comparison to prior work (Section 6) and some concluding remarks (Section 7).

2 Background

Symbolic model checking [5, 7, 11] is a fully automatic verification paradigm that checks temporal properties (e.g., safety, liveness, fairness, etc.) of finite state systems by symbolic state traversal. The core enabling technology for symbolic model checking is the use of the Binary Decision Diagram (BDD) representation [4] for state sets and state transitions. BDDs represent Boolean formulas canonically as directed acyclic graphs such that equivalent sub-formulas are uniquely represented as a single subgraph. This uniqueness property makes BDDs compact and enables dynamic programming to be used for computing Boolean operations symbolically.

To use BDDs in model checking, we need to map sets of states, state transitions, and state traversal to the Boolean domain. In this section, we briefly describe this mapping and motivate how time-invariant constraints arise. We finish with definitions of some additional terminology to be used in the rest of the paper.

2.1 Representing State Sets and Transitions

In the symbolic model checking of finite state systems, a state typically describes the values of many components (e.g., latches in digital circuits) and each component is represented by a *state variable*. Let $V = \{v_1, ..., v_n\}$ be the set of state variables in a system, then a state can be described by assigning values to all the variables in V. This valuation can in term be written as a Boolean formula that is true exactly for the valuation as $\bigwedge_{i=0}^{n}(v_i == c_i)$, where c_i is the value assigned to the variable v_i, and the "==" represents the equality operator in a predicate (similar to the C programming language). A set of states can be represented as a disjunction of the Boolean formulas that represent the states. We denote the BDD representation for a set of states S by $S(V)$.

In addition to the set of states, we also need to map the system's state transitions to the Boolean domain. We extend the above concept of representing a set of states to representing a set of ordered-pairs of states. To represent a pair of states, we need two sets of state variables: V the set of *present-state variables* for the first tuple and V' the set of *next-state variables* for the second tuple. Each variable v in V has a corresponding next-state variable v' in V'. A valuation of variables in V and V' can be viewed as a state transition from one state to another. A transition relation can then be represented as a set of these valuations. We denote the BDD representation of a transition relation T as $T(V, V')$.

In modeling finite state systems, the overall state transitions are generally specified by defining the valid transitions for each state variable. To support non-deterministic transitions of a state variable, the expression that defines the transitions evaluates to a set, and the next-state value of the state variable is non-deterministically chosen from the elements in the set. Hereafter, we refer to an expression that evaluates to a set either as a *set expression* or as a *non-deterministic expression* depending on the context, and we use the bold font type, as in **f**, to represent such expression. Let $\mathbf{f}_i$ be the set expression representing state transitions of the state variable v_i. Then the BDD representation for v_i's transition relation T_i

can be defined as

$$T_i(V, V') := (v'_i \in \mathbf{f}_i(V)).$$

For synchronous systems, the BDD for the overall state transition relation T is

$$T(V, V') := \bigwedge_{i=0}^{n} T_i(V, V').$$

Detailed descriptions on this formulation, including mapping of asynchronous systems, can be found in [5, 11].

Note that in the above formulation, we also need to represent non-Boolean expressions (such as $\mathbf{f}_i$'s) or non-Boolean variables as part of intermediate step. These non-Boolean formulas can be represented using variants of BDDs (e.g., MTBDD [6]) that extend the BDD concept to include non-Boolean values. These extensions also include algorithms for Boolean operators with non-Boolean arguments like the "==" and "$\in$". We will refer to all these BDD-variants simply as BDDs.

2.2 Time-Invariant Constraints and Their Common Usages

In symbolic model checking, time-invariant constraints specify the conditions that must always hold. More formally, let $C_1, \ldots, C_l$ be the time-invariant constraints and let $C := C_1 \wedge C_2 \wedge \ldots \wedge C_l$. Then, in symbolic state traversal, we consider only states where C is true. We refer to C as the *constrained space*.

To motivate how time-invariant constraints arise naturally in modeling complex systems, we describe three common usages. One common usage is to make the same non-deterministic choice across multiple expressions in transition relations. For example, in a master-slave model, the master can non-deterministically choose which set of idle slaves to assign the pending jobs, and the slaves' next-state values will depend on the choice made. To model this, let $\mathbf{f}$ be the non-deterministic expression representing how the master makes its choice. If the slaves' transition relations are defined using the expression $\mathbf{f}$ directly, then each use of $\mathbf{f}$ makes its own non-deterministic choice independent of other uses. Thus, to ensure that all the slaves see the same non-deterministic choice, a new state variable u is introduced to record the choice made, and u is then used to define the slaves' transition relations. This recording process is expressed as the time-invariant constraint $u \in \mathbf{f}$.

Another common usage is for establishing the interface between different components in a system. For example, suppose two components are connected with a pipe of a fixed capacity. Then, the input of one component is the minimum of the pipe's capacity and the output of the other component. This relationship is described as a time-invariant constraint between the input and the output of these two components.

Third common usage is specific uses of generic parts. For example, a bi-directional fuel pipe may be used to connect two components. If we want to make sure the fuel flows only one way, we need to constrain the valves in the fuel pipe. These constraints are specified as time-invariant constraints. In general, specific uses of generic parts arise naturally in both the software and the hardware domain as we often use generic building blocks in constructing a complex system.

In the examples above, the use of time-invariant constraints is not always necessary because some these constraints can be directly expressed as a part of the transition relation and the associated state variables can be removed. However, these constraints are used to facilitate the description of the system or to reflect the way complex systems are built. Without these constraints, multiple expressions will need to be combined into possibly a very complicated expression. Performing this transformation manually can be labor intensive and error-prone. Thus it is up to the verification tool to automatically perform these transformations and remove unnecessary state variables. Our optimizations for constraint-rich models is to automatically eliminate redundant state variables (Section 4) and partition the remaining constraints (Section 3).

2.3 Symbolic State Traversal

To reason about temporal properties, the *pre-image* and the *image* of the transition relation are used for symbolic state traversal, and time-invariant constraints are used to restrict the valid state space. Based on the BDD representations of a state set S and the transition relation T, we can compute the *pre-image* and the *image* of S, while restricting the computations to the constrained space C, as follows:

$$\begin{aligned} \textit{pre-image}(S)(V) &:= C(V) \wedge \exists V'.[T(V,V') \wedge (S(V') \wedge C(V'))] \quad (1) \\ \textit{image}(S)(V') &:= C(V') \wedge \exists V.[T(V,V') \wedge (S(V) \wedge C(V))] \quad (2) \end{aligned}$$

One limitation of the BDD representation is that the monolithic BDD for the transition relation T is often too large to build. A solution to this problem is the *conjunctive partitioning* [5] of the transition relation. In conjunctive partitioning, the transition relation is represented as a conjunction $P_1 \wedge P_2 \wedge ... \wedge P_k$ with each conjunct P_i represented by a BDD. Then, the pre-image can be computed by conjuncting with one P_i at a time, and by using *early quantification* to quantify out variables as soon as possible. The early-quantification optimization is based on the property that sub-formulas can be moved out of the scope of an existential quantification if they do not depend on any of the variables being quantified. Formally, let V_i', a subset of V', be the set of variables that do not appear in any of the subsequent P_j's, where $1 \leq i \leq k$ and $i < j \leq k$. Then the pre-image can be computed as

$$
\begin{aligned}
p_1 &:= \exists V_1'.[P_1(V,V') \wedge (S(V') \wedge C(V'))] \qquad (3)\\
p_2 &:= \exists V_2'.[P_2(V,V') \wedge p_1]\\
&\vdots\\
p_k &:= \exists V_k'.[P_k(V,V') \wedge p_{k-1}]\\
\textit{pre-image}(S)(V) &:= C(V) \wedge p_k
\end{aligned}
$$

The determination and ordering of partitions (the P_i's in above) can have significant performance impact. Commonly used heuristics [8, 12] treat the state variables' transition relations (T_i's) as the partitions. The ordering step then greedily schedules the partitions to quantify out more variables as soon as possible, while introducing fewer new variables. Finally, the ordered partitions are tentatively merged with their predecessors to reduce the number of intermediate results. Each merged result is kept only if the resulting graph size is less than a pre-determined limit.

The conjunctive partitioning for the image computation is performed similarly with present-state variables in V being the quantifying variables instead of next-state variables in V'. However, since the quantifying variables are different between the image and the pre-image computation, the resulting conjuncts for image computation is typically very different from those for pre-image computation.

2.4 Additional Terminology

We define the *support variables* of a function to be the variables that the function depends on, and the *support* of a function to be the set of support variables. We define the *ITE* operator (if-then-else) as follows: given arbitrary expressions f and g where f and g may both be set expressions, and Boolean expression p, then

$$\textit{ITE}(p,f,g)(X) := \begin{cases} f(X) & \text{if } p(X); \\ g(X) & \text{otherwise;} \end{cases}$$

where X is the set of variables used in expressions p, f, and g. We define a *care-space optimization* as any algorithm *care-opt* that has following properties: given an arbitrary expression f where f may be a set expression, and a Boolean formula c, then

$$\textit{care-opt}(f,c) := \textit{ITE}(c,f,d),$$

where d is defined by the particular algorithm used. The usual interpretation of this is that we only *care* about the values of f when c is true. We will refer to c as the *care space* and $\neg c$ as the *don't-care space*. The goal of care-space optimizations is to heuristically minimize the representation for f by choosing a suitable d in the don't-care space. Descriptions and a study of some care-space optimizations, including the commonly used *restrict* algorithm [7], can be found in [14].

3 Extended Conjunctive Partitioning

The first optimization is the application of the conjunctive-partitioning algorithm on the time-invariant constraints. This extension is derived based on two observations. First, as with the transition relations, the BDD representation for time-invariant constraints can be too large to be represented as a monolithic graph. Thus, it is crucial to represent the constraints as a set of conjuncts rather than a monolithic graph.

Second, in constraint-rich models, many *quantifying variables* (variables being quantified) do not appear in the transition relation. There are two common causes for this. First, when time-invariant constraints are used to make the same non-deterministic choices, new variables are introduced to record these choices (described as the first example in Section 2.2). In the transition relation, these new variables are used only in their present-state form. Thus, their corresponding next-state variables do not appear in the transition relation, and for the pre-image computation, these next-state variables are parts of the quantifying variables. The other cause is that many state variables are used only to establish time-invariant constraints. Thus, both the present- and the next-state version of these variables do not appear in the transition relations.

Based on this observation, we can improve the early-quantification optimization by pulling out the quantifying variables (V'_0) that do not appear in any of the transition relations. Then, these quantifying variables (V'_0) can be used for early quantification in conjunctive partitioning of the constrained space (C) where the time-invariant constraints hold. Formally, let $Q_1, Q_2, ..., Q_m$ be the partitions produced by the conjunctive partitioning of the constrained space C, where $C = Q_1 \wedge Q_2 \wedge ... \wedge Q_m$. For the pre-image computation, Equation 3 is replaced by

$$\begin{aligned}
q_1 &:= \exists W'_1.[Q_1(V') \wedge S(V')] \\
q_2 &:= \exists W'_2.[Q_2(V') \wedge q_1] \\
&\ \vdots \\
q_m &:= \exists W'_m.[Q_m(V') \wedge q_{m-1}] \\
p_1 &:= \exists V'_1.[P_1(V, V') \wedge q_m]
\end{aligned}$$

where W_i', a subset of V_0', is the set of variables that do not appear in any of the subsequent Q_j's, where $1 \leq i \leq m$ and $i < j \leq m$. Similarly, this extension also applies to the image computation.

4 Elimination of Redundant State Variables

Our second optimization for constraint-rich models is targeted at reducing the state space by removing unnecessary state variables. This optimization is a set of BDD-based algorithms that compute an equivalent expression for each variable used in the time-invariant constraints (*macro extraction*) and then globally replace a suitable subset of variables with their equivalent expressions (*macro expansion*) to reduce the total number of variables.

The use of macros is traditionally supported by language constructs (e.g., DEFINE in the SMV language [11]) and by simple syntactic analyses such as detecting deterministic assignments (e.g., $a == f$ where a is a state variable and f is an expression) in the specifications. However, in constraint-rich models, the constraints are often specified in a more complex manner such as *conditional* dependencies on other state variables (e.g., $p \Rightarrow (a == f)$ as conditional assignment of expression f to variable a when p is true). To identify the set of valid macros in such models, we need to combine the effects of multiple constraints. One drawback of syntactic analysis is that, for each type of expression, syntactic analysis will need to add a template to pattern match these expressions. Another more severe drawback is that it is difficult for syntactic analysis to estimate the actual cost of instantiating a macro. Estimating this cost is important because reducing the number of variables by macro expansion can sometimes result in significant performance degradation caused by large increases in other BDD sizes. These two drawbacks make the syntactic approach unsuitable for models with complex time-invariant constraints.

Our approach uses BDD-based algorithms to analyze time-invariant constraints and to derive the set of possible macros. The core algorithm is a new assignment-extraction algorithm that extracts assignments from arbitrary Boolean expressions (Section 4.1). For each variable, by extracting its assignment form, we can determine the variable's corresponding equivalent expression, and when appropriate, globally replace the variable with its equivalent expression (Section 4.2). The strength of this algorithm is that by using BDDs, the cost of macro expansion can be better characterized because the actual model checking computation is performed using BDDs.

Note that there have been a number of research efforts on BDD-based redundant state-variable removal. To better compare our approach to these previous research efforts, we postpone the discussion of this prior work until Section 6, after describing our algorithms and the performance evaluation.

4.1 BDD-Based Assignment Extraction

The assignment-extraction problem can be stated as follows: given an arbitrary Boolean formula f and a variable v (where v can be non-Boolean), find $\mathbf{g}$ and h such that

- $f = (v \in \mathbf{g}) \wedge h$,
- $\mathbf{g}$ does not depend on v, and
- h is a Boolean formula and does not depend on v.

The expression $(v \in \mathbf{g})$ represents a non-deterministic assignment to variable v. In the case that $\mathbf{g}$ always evaluates to a singleton set, the assignment $(v \in \mathbf{g})$ is deterministic. A solution to this assignment-extraction problem is as follows:

$$\begin{aligned} h &= \exists v . f \\ \mathbf{t} &= \bigcup_{k \in K_v} \mathit{ITE}(f|_{v \leftarrow k}, \{k\}, \emptyset) \\ \mathbf{g} &= \mathit{restrict}(\mathbf{t}, h) \end{aligned}$$

where K_v is the set of all possible values of variable v, and *restrict* [7] is a care-space optimization algorithm that tries to reduce the BDD graph size (of $\mathbf{t}$) by collapsing the don't-care space $(\neg h)$. The BDD algorithm for the $\bigcup_{k \in K_v}$ operator is similar to the BDD algorithm for the existential quantification with the $\vee$ operator replaced by the $\cup$ operator for variable quantification. A correctness proof of this algorithm is included in Appendix A.

Currently, we do not have any optimality guarantees for this solution. We do know that the solution $\mathbf{g}$ does *not* always have the minimum number of support variables. i.e., $\mathbf{g}$ is not necessary a *minimum-support* solution. However, based on the behavior of the *restrict* algorithm, we have formed a conjecture about the relationship between $\mathbf{g}$ and any minimum-support solution. Informally, this conjecture states that $\mathbf{g}$ can be converted to any minimum-support solution via a sequence of variable substitutions, and that these substitutions are parts of equality decomposition of h. More formally,

Conjecture 1 *Let $\mathbf{g}_{min}$ be a solution with the minimum number of support variables. Then, there exists a sequence of n substitutions $(u_i \leftarrow e_i)$ where u_i's are variables, e_i's are expressions, and $1 \leq i \leq n$, such that*

- *let $\mathbf{g}_0 = \mathbf{g}$, and $\mathbf{g}_i = \mathbf{g}_{i-1}|_{u_i \leftarrow e_i}$ for $1 \leq i \leq n$, then $\mathbf{g}_n = \mathbf{g}_{min}$, and*

- $h \Rightarrow (\bigwedge_{i=1}^{n}(u_i == e_i))$.

If this conjecture is true, then we know that even though each right-hand-side expression **g** extracted might have more support variables than necessary, these additional variables can be eliminated later by other macros (the $(u_i == e_i)$'s above).

The above conjecture is based on the observation that given variable u, expressions f and e, where

- f depends on u,
- e does not depend on u, and
- u's variable order succeeds any of e's support variables' variable order,

then, $restrict(f, (u == e)) = f|_{u \leftarrow e}$. Another way of looking at this is that using *restrict*, a variable (u) might not be removed if its variable order does not come after the orders of its equivalent expression's (e's) support variables. To prove the above conjecture, we will need to first prove that equality expressions are the only reason that the *restrict* algorithm may not produce minimum support solution.

Note that in the assignment-extraction algorithm, the use of the *restrict* algorithm is not necessary. In fact, any care-space optimization algorithms can be used instead of the *restrict* algorithm. We choose to use the *restrict* algorithm because of the above property and because it works well in practice for other symbolic-model-checking computations.

4.2 Macro Extraction and Expansion

In this section, we describe the elimination of state variables based on macro extraction and macro expansion. The first step is to extract macros with the algorithm shown in Figure 1. This algorithm extracts macros from the constrained space (C), which is represented as a set of conjuncts. It first uses the assignment-extraction algorithm to extract assignment expressions (line 5). It then identifies the deterministic assignments as candidate macros (line 6). For each candidate, the algorithm tests to see if applying the macro may be beneficial (line 7). This test is based on the heuristic that if the BDD graph size of a macro is not too large and its instantiation does not cause excessive increase in other BDDs' graph sizes, then instantiating this macro may be beneficial. If the resulting right-hand-side **g** is not a singleton set, it is kept separately (line 9). These **g**'s are combined later (line 10) to determine if their intersection would result in a macro (lines 11-13). Finally, this algorithm returns the set of selected macros (line 14).

After the macros are extracted, the next step is to determine the instantiation order. The main purpose of this algorithm (in Figure 2) is to remove circular dependencies. For example, if one macro defines variable v_1 to be $(v_2 \wedge v_3)$ and a second

```
   extract_macros($C$, $V$)
     /* Extract macros for variables in $V$ from
        the set $C$ of conjuncts representing the constrained space */
1    $M \leftarrow \emptyset$   /* initialize the set of macros found so far */
2    for each $v \in V$
3       $N \leftarrow \emptyset$   /* initialize the set of non-singletons found so far */
4       for each $f \in C$ such that $f$ depends on $v$
5          $(\mathbf{g}, h) \leftarrow$ assignment-extraction $(f, v)$   /* $f = (v \in \mathbf{g}) \wedge h$ */
6          if ($\mathbf{g}$ always returns a singleton set)   /* macro found */
7             if (is-this-result-good($\mathbf{g}$))
8                $M \leftarrow \{(v, \mathbf{g})\} \cup M$
9          else $N \leftarrow \{\mathbf{g}\} \cup N$
10      $\mathbf{g'} \leftarrow \bigcap_{\mathbf{g} \in N} \mathbf{g}$
11      if ($\mathbf{g'}$ always returns a singleton set)   /* macro found */
12         if ((is-this-result-good($\mathbf{g'}$))
13            $M \leftarrow \{(v, \mathbf{g'})\} \cup M$
14   return $M$
```

Figure 1: Macro-extraction algorithm. In lines 7 and 12, "is-this-result-good" uses BDD properties (such as graph sizes) to determine if the result should be kept.

macro defines v_2 to be $(v_1 \vee v_4)$, then instantiating the first macro results in a circular definition in the second macro ($v_2 = (v_2 \wedge v_3) \vee v_4$) and thus invalidates this second macro. Similarly, the reverse is also true. To determine the set of macros to remove, the algorithm builds a dependence graph (line 1) and breaks circular dependencies based on graph sizes (lines 2-4). It then determines the ordering of the remaining macros based on the topological order (line 4) of the dependence graph.

Finally, in the topological order, each macro $(v, \mathbf{g})$ is instantiated in the remaining macros and in all other expressions (represented by BDDs) in the system, by substituting the variable v with its equivalent expression $\mathbf{g}$.

5 Evaluation

5.1 Experimental Setup

The benchmark suite used is a collection of 58 SMV models gathered from a wide variety of sources, including the 16 models used in a BDD performance study [17]. Out of these 58 models, 37 models have no time-invariant constraints, and thus our optimizations are not triggered and have no influence on the overall verification time. Out of the remaining 21 models, 10 very small models ($<$ 10 seconds)

order_macros(M)
/* Determine the instantiation order of the macros in set M */
/* first build the dependence graph $G = (M, E)$ */
$E = \{(x, y) | x = (v_x, \mathbf{g}_x) \in M, y = (v_y, \mathbf{g}_y) \in M, \mathbf{g}_y$ depends on $v_x\}$
/* then remove circular dependences */
while there are cycles in G,
  $M_C \leftarrow$ set of macros that are in some cycle
  remove the macro with largest BDD size in M_C
return a topological ordering of the remaining macros in G

Figure 2: Macro-ordering algorithm.

are eliminated. On the remaining 11 models, our optimizations have made non-negligible performance impact on 7 models, where the results changed by more than 10 CPU seconds and 10% from the base case where no optimizations are enabled. In Figure 3, we briefly describe these 7 models. Note that some of these models are quite large, with up to 1200 state bits.

Model	# of State Bits	Description
acs	497	the altitude-control module of NASA's DS1 spacecraft
ds1-b	657	a buggy fault diagnosis model for NASA's DS1 spacecraft
ds1	657	corrected version of *ds1-b*
futurebus	174	FutureBus cache coherency protocol
nomad	1273	fault diagnosis model for an Antarctic meteorite explorer
v-gate	86	reactor-system model
xavier	100	fault diagnosis model for the Xavier robot

Figure 3: Description of models whose performance results are affected by our optimizations.

The results reported in this section are labeled with the following keys to indicate which optimizations are enabled:

None: no optimizations.

Quan: the "early quantification on the constrained space" optimization (Section 3).

SynM: syntactic analysis for macro-extraction and macro-expansion. This algorithm pattern matches deterministic assignment expressions ($v == f$, where v is a state variable and f is an expression) as macros and expands these macros.

BDDM: the BDD-based macro extraction and macro expansion (Section 4).

Q+SynM: both **Quan** and **SynM** optimizations.

Q+BDDM: both **Quan** and **BDDM** optimizations.

We performed the evaluation using the Symbolic Model Verifier (SMV) model checker [11] from Carnegie Mellon University. Conjunctive partitioning was used only when it was necessary to complete the verification. In these cases (including *acs, nomad, ds1-b,* and *ds1*), the size limit for each partition was set to 10,000 BDD nodes. For the remaining cases, the transition relations were represented as monolithic BDDs. The constrained space C was represented as a conjunction with each conjunct's BDD graph size limited to 10,000 nodes. Without partitioning, we could not construct the BDD representation for the constrained space for 4 models. The evaluation was performed on a 200MHz Pentium-Pro with 1 GB of memory running Linux. Each run was limited to 6 hours of CPU time and 900 MB of memory.

In Figure 4, we show the running time of different optimizations. Note that for all benchmarks, the time spent by our optimizations is very small (< 5 seconds or $< 5\%$ of total time) and is included in the running time shown. In the rest of this section, we analyze these results in the following order: the overall impact of our optimizations (Section 5.2), the impact of early quantification on the constraint space (Section 5.3), and the impact of macro optimization (Section 5.4). We then finish with a brief study on the impact of different size limits for conjunctive partitioning (Section 5.5).

Model	**None** (sec)	**Quan** (sec)	**SynM** (sec)	**BDDM** (sec)	**Q+SynM** (sec)	**Q+BDDM** (sec)
acs	*m.o.*	32	*m.o.*	1059	76	7
ds1-b	*m.o.*	321	*t.o.*	*m.o.*	138	54
ds1	*m.o.*	*m.o.*	*m.o.*	*t.o.*	*t.o.*	37
futurebus	1410	53	78	37	35	19
nomad	*m.o.*	*t.o.*	*m.o.*	*t.o.*	7801	633
v-gates	36	35	51	50	53	50
xavier	16	5	6	5	1	2

Figure 4: Running time with different optimizations enabled. The *m.o.*'s and *t.o.*'s are the results that exceeded the 900-MB memory limit and the 6-hour time limit, respectively.

5.2 Overall Results

The results in Figure 5 show the overall performance impact of our optimizations. These results demonstrate that our optimizations have significantly improved the performance for 2 models (with speedups up to 74) and have enabled the verification of 4 models. For the *v-gates* model, the performance degradation (speedup = 0.7) is in the computation of the reachable states from the initial states. Upon further investigation, we believe that it is caused by the macro optimization, which increases the graph size of the transition relation from 122-thousand to 476-thousand nodes. This case demonstrates that reducing the number of state variables does not always improve performance.

Model	**None** (sec)	**Q+BDDM** (sec)	**None / Q+BDDM** (speedup)
acs	*m.o.*	7	*enabled*
ds1-buggy	*m.o.*	54	*enabled*
ds1	*m.o.*	367	*enabled*
futurebus	1410	19	74.2
nomad	*m.o.*	633	*enabled*
valves-gates	36	50	0.7
xavier	16	2	8.0

Figure 5: Overall impact of our optimizations. The *m.o.*'s are the results that exceeded the 900-MB memory limit.

5.3 Impact of Early Quantification

The results in Figure 6 show the impact of applying early quantification on time-invariant constraints. The impact is measured both in the number of quantifying BDD variables extracted from the transition relations and in the performance speedups. The speedup results for **None / Quan** show that adding this optimization has enabled the verification of *acs* and *ds1-b*, and achieved significant performance improvement on *futurebus* (speedup of 26). The results in the **Quan** columns show that this improvement is mostly due to the fact that a large number of variables can be pulled out of the transition relations and applied to conjunctive partitioning and early quantification of the time-invariant constraints.

From the **Q+BDDM** and **BDDM / Q+BDDM** columns, we observe similar results in presence of BDD-based macro optimization. Note that for the **Q+BDDM** columns, the "# of BDD vars extracted" results also include the number of BDD variables that are removed by the macro optimization. This is done to make the comparison between **Quan** and **Q+BDDM** results easier.

The results in Figure 6 show two additional interesting points. First, the number of variables extracted for pre-image computation is more than that extracted for image computation. This is because some variables are only used in their present-state form in the transition relation (see first example in Section 2.2). Second, comparing the results between **Quan** and **Q+BDDM** columns indicates that the macro optimization generally does not interfere with the early-quantification optimization. The one exception is the *nomad* model, where the macro optimization introduced 114 BDD variables ((1121 - 1067) present-state variables plus (1174 - 1114) next-state variables) to the overall transition relation.

	Total # of BDD Vars	**Effects of CP Optimization**					
		# of BDD vars extracted				performance speedup	
		Quan		**Q+BDDM**		**None / Quan**	**BDDM / Q+BDDM**
Model		img	p-img	img	p-img		
acs	994	439	449	437	449	*enabled*	151.0
ds1-b	1314	550	566	546	566	*enabled*	*enabled*
ds1	1314	550	566	546	566	n/a	*enabled*
futurebus	348	58	110	54	110	26.6	1.9
nomad	2546	1121	1174	1067	1114	*n/a*	*enabled*
v-gates	172	0	17	8	17	1.0	1.0
xavier	200	69	86	69	86	3.2	2.5

Figure 6: Effectiveness of the extended conjunctive-partitioning optimization. The effectiveness measures are (1) the number of quantifying BDD variables that are pulled out of the transition relation for early quantification of the time-invariant constraints, and (2) the impact on overall running time as performance speedups. For both measures, we present results both with (**+BDDM**) and without the BDD-based macro optimization. The *n/a* indicates that the speedup can not be computed because both cases failed to finish within the resource limits. **Note:** the number of BDD variables is twice the number of state variables—one copy for the present state and one copy for the next state.

5.4 Impact of Macro Extraction and Macro Expansion

The results in Figure 7 show the impact of the BDD-based macro optimization, This impact is measured both in the number of BDD variables removed and in the performance speedups. The performance results in the **None / BDDM** column show that adding this optimization has enabled the verification of *acs* and achieved significant performance improvement on *futurebus* (speedup of 38). The results are similar in presence of the early-quantification optimization (the **Quan / Q+BDDM** column). The results for "# of BDD vars removed" show that these performance improvements are due to the effectiveness of BDD-based macro optimization in removing variables; in particular, over a third of variables are removed for 4 models.

Model	Total # of BDD Variables	Effects of BDD-based Macro Optimization		
		# of BDD variables removed	performance speedup	
			None / BDDM	**Quan / Q+BDDM**
acs	994	352	*enabled*	4.5
ds1-b	1314	492	*n/a*	5.9
ds1	1314	496	*n/a*	*enabled*
futurebus	348	18	38.1	2.7
nomad	2546	844	*n/a*	*enabled*
v-gates	172	16	0.7	0.7
xavier	200	116	3.2	2.5

Figure 7: Effectiveness of macro optimizations. The effectiveness measures are (1) the number of BDD variables removed by macro optimization, and (2) the impact on overall running time as performance speedups. For both measures, we present results both with and without the early-quantification optimization. The *n/a* indicates that the speedup can not be computed because both cases failed to finish within the resource limits. **Note:** the number of BDD variables is twice the number of state variables.

To evaluate the effectiveness of syntactic-based vs. BDD-based macro extraction, we compare the impact of these two approaches using both the number of BDD variables removed and the running time (Figure 8). The comparison is done for both with and without the early-quantification optimization. Note that the early-quantification optimization does not affect the number of BDD variables removed. Thus, the "# of BDD vars removed" results are the same for both with and without the early-quantification optimization.

Without the early-quantification optimization (the **SynM / BDDM** column), the results show that the BDD-based approach is better with the verification of *acs* enabled. With the early-quantification optimization (the **Q+SynM / Q+BDDM** column), the results show that the BDD-based approach has enabled the verification of *ds1* and generally has better performance, with speedups of over 10 in *acs* and *nomad*. In the *xavier* case, the slowdown is 2 (speedup of 0.5) because the **Q+BDDM** used one extra second in macro extraction. The overall performance improvements are due to that the BDD-based approach is more effective in reducing the number of variables ("# of BDD vars removed" columns). In particular, for the *acs, nomad, ds1-b, and ds1* models, > 150 additional BDD variables (i.e.,

> 75 state bits) are removed in comparison to using syntactic analysis.

Model	Total # of BDD Vars	Syntax vs. BDD-based Macro Optimization			
		# of BDD vars removed		performance speedup	
		SynM or **Q+SynM**	**BDDM** or **Q+BDDM**	**SynM / BDDM**	**Q+SynM / Q+BDDM**
acs	994	82	352	*enabled*	10.8
ds1-b	1314	148	492	*n/a*	2.5
ds1	1314	220	496	*n/a*	*enabled*
futurebus	348	12	18	2.1	1.8
nomad	2546	688	844	*n/a*	12.3
v-gates	172	16	16	1.0	1.0
xavier	200	64	116	1.2	1 / 2 = 0.5

Figure 8: Syntactic-based vs. BDD-based macro optimization. The effectiveness measures are (1) the number of BDD variables removed, and (2) the impact on overall running time as performance speedups. For both measures, we present results both with and without the early-quantification optimization. The *n/a* indicates that the speedup can not be computed because both cases failed to finish within the resource limits. **Note:** the number of BDD variables is twice the number of state variables.

5.5 Impact of Conjunctive-Partitioning Size Limit

Because the conjunctive-partitioning algorithm often produces significantly different performance results with different partition-size limits, we have also re-evaluated the above results using a partition-size limit of 100,000 nodes. The new results generally follow the same trend as before with the exception of *ds1-b* and *ds1*. For these two models, the results (Figure 9) show that if we choose the right partition-size limit for each case, we do not need to perform the macro optimization to verify them. (Note that this is not always true; e.g., the *nomad* model cannot be verified without the macro optimization.) However, with the BDD-based macro optimization (the **Q+BDDM** column), the performance results are more stable and are generally much better.

6 Related Work

There have been many research efforts on BDD-based redundant state-variable removal in both logic synthesis and verification. These research efforts all use

Model	Partition Size Limit (# of nodes)	**None** (sec)	**Quan** (sec)	**SynM** (sec)	**BDDM** (sec)	**Q+SynM** (sec)	**Q+BDDM** (sec)
ds1-b	10,000	*m.o.*	321	*t.o.*	*m.o.*	138	54
ds1-b	100,000	*t.o.*	309	*t.o.*	*m.o.*	*t.o.*	101
ds1	10,000	*m.o.*	*m.o.*	*m.o.*	*t.o.*	*t.o.*	37
ds1	100,000	*m.o.*	255	*t.o.*	*m.o.*	92	74

Figure 9: Effects of the partition-size limit. The *m.o.*'s and *t.o.*'s are the results that exceeded the 900-MB memory limit and the 6-hour time limit, respectively.

the reachable state space (set of states reachable from initial states) to determine functional dependencies for Boolean variables (macro extraction). The reachable state space effectively plays the same role as a time-invariant constraint, because the verification process only needs to check specifications in the reachable state space.

Berthet et al. propose the first redundant state-variable removal algorithm in [3]. In [10], Lin and Newton describe a branch-and-bound algorithm to identify the maximum set of redundant state variables. In [13], Sentovich et al. propose new algorithms for latch removal and latch replacement in logic synthesis. There is also some work on detecting and removing redundant state variables while the reachable state space is being computed [9, 15].

From the algorithmic point of view, our approach is different from prior work in two ways. First, in determining the relationship between variables, the algorithms used to extract functional dependencies in previous work can be viewed as direct extraction of deterministic assignments to Boolean variables. In comparison, our assignment extraction algorithm is more general because it can also handle non-Boolean variables and extract non-deterministic assignments. Second, in performing the redundant state-variable removal, the approach used in the previous work would need to combine all the constraints first and then extract the macros directly from the combined result. However, for constraint-rich models, it may not be possible to combine all the constraints because the resulting BDD is too large to build. Our approach addresses this issue by first applying the assignment extraction algorithm to each constraint separately and then combining the results to determine if a macro can be extracted (see Figure 1).

Another difference is that in previous work, the goal is to remove as many variables as possible. However, we have empirically observed that in some cases, removing additional variables can result in significant performance degradation in overall verification time (slowdown over 4). To address this issue, we use simple heuristics (size of the macro and the growth in graph sizes) to choose the set of macros to expand. This simple heuristic works well in the test cases we tried. However, in order to fully evaluate the impact of different heuristics, we need to gather a larger set of constraint-rich models from a wider range of applications.

7 Conclusions and Future Work

The two optimizations we proposed are crucial in verifying this new class of constraint-rich applications. In particular, they have enabled the verification of real-world applications such as the Nomad robot and the NASA Deep Space One spacecraft.

We have shown that the BDD-based assignment-extraction algorithm is effective in identifying macros. We plan to use this algorithm to perform a more precise cone-of-influence analysis with the assignment expressions providing the exact dependence information between the variables. In general, we plan to study how BDDs can be use to further help other compile-time optimizations in symbolic model checking.

Acknowledgement

We thank Ken McMillan for discussions on the effects of macro expansion. We thank Olivier Coudert, Fabio Somenzi and reviewers for comments on this work. We are grateful to Intel Corporation for donating the machines used in this work.

A Correctness Proof for Assignment-Extraction Algorithm

In this section, we present a correctness proof for the assignment-extraction algorithm in Section 4.1. Before presenting the main result, we first state and prove two supporting lemmas.

Lemma 1 *Let care-opt be any care-space optimization. Then, for arbitrary set expression* $\mathbf{t}$*, Boolean formula* h*, and variable* v*,*

$$(v \in \textit{care-opt}(\mathbf{t}, h)) \wedge h = \textit{care-opt}(v \in \mathbf{t}, h) \wedge h.$$

Proof

By the definition of the care-space optimization, we have the following properties:

$$\begin{aligned} h &\Rightarrow (\textit{care-opt}(\mathbf{t}, h) == \mathbf{t}), \\ h &\Rightarrow [\textit{care-opt}(v \in \mathbf{t}, h) == (v \in \mathbf{t})]. \end{aligned}$$

Therefore,

$$\begin{aligned}(v \in \textit{care-opt}(\mathbf{t}, h)) \wedge h &= (v \in \mathbf{t}) \wedge h \\ &= \textit{care-opt}(v \in \mathbf{t}, h) \wedge h.\end{aligned}$$

$\square$

Lemma 2 *Given an arbitrary Boolean formula f and a variable v. Let*

$$\mathbf{t} = \bigcup_{k \in K_v} \textit{ITE}(f|_{v \leftarrow k}, \{k\}, \emptyset),$$

where K_v is the set of all possible values of variable v. Then,

$$(v \in \mathbf{t}) = f.$$

Proof

$$\begin{aligned}v \in \mathbf{t} &= \bigvee_{k' \in K_v} (v == k') \wedge (k' \in \mathbf{t}) \\ &= \bigvee_{k' \in K_v} (v == k') \wedge [k' \in \bigcup_{k \in K_v} \textit{ITE}(f|_{v \leftarrow k}, \{k\}, \emptyset)] \\ &= \bigvee_{k' \in K_v} (v == k') \wedge \bigvee_{k \in K_v} [k' \in \textit{ITE}(f|_{v \leftarrow k}, \{k\}, \emptyset)] \\ &= \bigvee_{k' \in K_v} (v == k') \wedge \bigvee_{k \in K_v} \textit{ITE}(f|_{v \leftarrow k}, k' \in \{k\}, k' \in \emptyset) \\ &= \bigvee_{k' \in K_v} (v == k') \wedge \bigvee_{k \in K_v} \textit{ITE}(f|_{v \leftarrow k}, k' \in \{k\}, 0) \\ &= \bigvee_{k' \in K_v} (v == k') \wedge \textit{ITE}(f|_{v \leftarrow k'}, k' \in \{k'\}, 0) \\ &= \bigvee_{k' \in K_v} (v == k') \wedge \textit{ITE}(f|_{v \leftarrow k'}, 1, 0) \\ &= \bigvee_{k' \in K_v} (v == k') \wedge f|_{v \leftarrow k'} \\ &= f.\end{aligned}$$

$\square$

Using the two lemmas above, we can now prove the correctness of the assignment-extraction algorithm.

Theorem 1 *Given an arbitrary Boolean formula f and a variable v. Let*

$$\begin{aligned} h &= \exists v.f, \\ \mathbf{t} &= \bigcup_{k \in K_v} \mathit{ITE}(f|_{v \leftarrow k}, \{k\}, \emptyset), \\ \mathbf{g} &= \mathit{care\text{-}opt}(\mathbf{t}, h), \end{aligned}$$

where care-opt is any care-space optimization algorithm and care-opt$(\mathbf{t}, h)$ does not depend on any new variables (other than those already in **t** *and h). Then, the following conditions are true.*

1. $f = (v \in \mathbf{g}) \wedge h$,
2. **g** *does not depend on v, and*
3. h *is a Boolean formula and does not depend on v.*

Proof

To prove Condition 1, we apply both Lemma 1 and Lemma 2 in the following derivation:

$$\begin{aligned} (v \in \mathbf{g}) \wedge h &= (v \in \mathit{care\text{-}opt}(\mathbf{t}, h)) \wedge h \\ &= \mathit{care\text{-}opt}(v \in \mathbf{t}, h) \wedge h \\ &= \mathit{care\text{-}opt}(f, h) \wedge h \\ &= f \wedge h \\ &= f \wedge \exists v.f \\ &= f \end{aligned}$$

Condition 2 is true because **t** does not depend on v (by construction) and the *care-opt* algorithm does not introduce new variable dependencies (given). Condition 3 is true because f is a Boolean formula and $h = \exists v.f$ is a Boolean formula that does not depend on v.

□

References

[1] Bapna, D., Rollins, E., Murphy, J., and Maimone, M. The Atacama Desert trek - outcomes. In *Proceedings of the 1998 International Conference on Robotics and Automation* (May 1998), pp. 597–604.

[2] BERNARD, D. E., DORAIS, G. A., FRY, C., JR., E. B. G., KANEFSKY, B., KURIEN, J., MILLAR, W., MUSCETTOLA, N., NAYAK, P. P., PELL, B., RAJAN, K., ROUQUETT, N., SMITH, B., AND WILLIAMS, B. Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the 1998 IEEE Aerospace Conference* (March 1998), pp. 259–281.

[3] BERTHET, C., COUDERT, O., AND MADRE, J. C. New ideas on symbolic manipulations of finite state machines. In *1990 IEEE Proceedings of the International Conference on Computer Design* (September 1990), pp. 224–227.

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

[5] BURCH, J. R., CLARKE, E. M., LONG, D. E., MCMILLAN, K. L., AND DILL, D. L. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13*, 4 (April 1994), 401–424.

[6] CLARKE, E. M., MCMILLAN, K. L., ZHAO, X., FUJITA, M., AND YANG, J. C.-Y. Spectral transform for large Boolean functions with application to technology mapping. In *Proceedings of the 30th ACM/IEEE Design Automation Conference* (June 1993), pp. 54–60.

[7] COUDERT, O., AND MADRE, J. C. A unified framework for the formal verification of circuits. In *Proceedings of the International Conference on Computer-Aided Design* (Feb 1990), pp. 126–129.

[8] GEIST, D., AND BEER, I. Efficient model checking by automated ordering of transition relation partitions. In *Proceedings of the Computer Aided Verification* (June 1994), pp. 299–310.

[9] HU, A. J., AND DILL, D. L. Reducing BDD size by exploiting functional dependencies. In *Proceedings of the 30th ACM/IEEE Design Automation Conference* (June 1993), pp. 266–71.

[10] LIN, B., AND NEWTON, A. R. Exact redundant state registers removal based on binary decision diagrams. *IFIP Transactions A, Computer Science and Technology A*, 1 (August 1991), 277–86.

[11] MCMILLAN, K. L. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[12] RANJAN, R. K., AZIZ, A., BRAYTON, R. K., PLESSIER, B., AND PIXLEY, C. Efficient BDD algorithms for FSM synthesis and verification. Presented in the IEEE/ACM International Workshop on Logic Synthesis, May 1995.

[13] SENTOVICH, E. M., AND HORIA TOMA, G. B. Latch optimization in circuits generated from high-level descriptions. In *Proceedings of the International Conference on Computer-Aided Design* (November 1996), pp. 428–35.

[14] SHIPLE, T. R., HOJATI, R., SANGIOVANNI-VINCENTELLI, A. L., AND BRAYTON, R. K. Heuristic minimization of BDDs using don't cares. In *Proceedings of the 31st ACM/IEEE Design Automation Conference* (June 1994), pp. 225–231.

[15] VAN EIJK, C. A. J., AND JESS, J. A. G. Exploiting functional dependencies in finite state machine verification. In *Proceedings of European Design and Test Conference* (March 1996), pp. 266–71.

[16] WILLIAMS, B. C., AND NAYAK, P. P. A model-based approach to reactive self-configuring systems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference* (August 1996), pp. 971–978.

[17] YANG, B., BRYANT, R. E., O'HALLARON, D. R., BIERE, A., COUDERT, O., JANSSEN, G., RANJAN, R. K., AND SOMENZI, F. A performance study of BDD-based model checking. In *Proceedings of the Formal Methods on Computer-Aided Design* (November 1998), pp. 255–289.