Symbolic Guided Search for CTL Model Checking*

Roderick Bloem[1] Kavita Ravi[2] Fabio Somenzi[1]

[1]University of Colorado
Boulder, CO
{Roderick.Bloem,Fabio}@Colorado.EDU

[2]Cadence Design Systems
New Providence, NJ
kravi@cadence.com

Abstract

CTL model checking of complex systems often suffers from the state-explosion problem. We propose using Symbolic Guided Search to avoid difficult-to-represent sections of the state space and prevent state explosion from occurring.

Symbolic Guided Search applies hints to guide the exploration of the state space. In this way, the size of the BDDs involved in the computation is controlled, and the truth of a property may be decided before all states have been explored. In this work, we show how hints can be used in the computation of nested fixpoints. We show how to use hints to obtain overapproximations useful for greatest fixpoints, and we present the first results for backward search. Our experiments demonstrate the effectiveness of our approach.

1 Introduction

Model checking is becoming a popular technique for debugging [Kur97]. This is especially true of circuits with complex control, for which it is often difficult to design simulation stimuli that will expose subtle bugs. On the other hand, the exhaustive verification afforded by model checking is often accompanied by the so-called state explosion problem. A combination of techniques, including the use of BDDs [Bry86, CBM89, McM94] or efficient SAT solvers [BCCZ99], the recourse to abstract interpretations [CC77] and assume-guarantee reasoning [CGL92], have been employed to mitigate the problem. The purpose of these techniques is either to reduce the complexity of the model subjected to verification, or to increase the size of the problems that can be successfully analyzed without further abstraction or decomposition. The most effective model checking approaches combine both types of attack to the state explosion problem.

Symbolic guided search has been recently proposed as a way to reduce the time and memory requirements of BDD-based invariant checking [RS99b, Rav99] and LTL model checking [BRS99]. Symbolic guided search is based on the observation that breadth-first search exploration of the state space of the model is often inefficient because it needs to represent sets of states that have no compact BDDs. It is indeed often the case that the fixpoints computed by model checking algorithms have BDDs of moderate or even small size, while the intermediate steps of the computations produce inordinately large diagrams.

Symbolic guided search uses hints, which are assertions on the primary inputs and state variables of the model, to direct the exploration of the state space. Given enough time and memory, the entire state space is explored, or, in other words, the fixpoints are computed exactly. However, by disabling problematic sets of transitions, and by visiting regularly arranged sets of states, guided search may dramatically improve both runtime and memory requirements. This is illustrated in Figure 3, which shows that the application of hints reduces the BDD sizes for the intermediate results in a CTL model checking experiment by over one order of magnitude.

The main contribution of this paper is the extension of symbolic guided search to CTL model checking. CTL uses backward search and both greatest and least fixpoint computations to decide the truth of a formula. The experimental results presented in [RS99b, BRS99] are for non-nested least fixpoint computation based on forward traversal. In this work we extend guided search to allow for nested fixpoints, and we introduce the use of hints to obtain overapproximations which is essential for efficient evaluation of greatest fixpoints. We also discuss incremental model checking as an alternative to overapproximations. We present the first results for backward state space traversal. To tackle nested fixpoints, we introduce two approaches based on *local* and *global* application of hints. The global approach is only applicable to the ECTL/ACTL fragment of the logic. We present a theory of symbolic guided search that encompasses not only the results of [RS99b, BRS99], but also those of [CCLQ97, NIJ+97]. In our experiments we analyze the effects of the application of hints and show that concrete advantages derive from the deployment of the technique we propose.

The rest of this paper is organized as follows. In Section 2 we recall basic concepts and introduce the notation used in the sequel. In Section 3 we extend symbolic guided search to CTL model checking. Section 4 is devoted to the experimental results, while Section 5 presents conclusions and discusses future work.

2 Preliminaries

The logic CTL [CE81] is defined over an alphabet A of atomic propositions: Any atomic proposition is a CTL property, and if φ and ψ are CTL properties, then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$, and $\mathsf{E}\varphi\,\mathsf{U}\,\psi$, $\mathsf{EG}\varphi$, and $\mathsf{EX}\varphi$. The semantics of CTL are defined over a Kripke structure $K = \langle S, T, S_0, A, L\rangle$, where S is the set of states, $T \subseteq S \times S$ is the transition relation, $S_0 \subseteq S$ is the set of initial states, A is the set of atomic propositions, and $L : S \to 2^A$ is the labeling function. The semantics of CTL are defined in Figure 1. If fairness constraints are specified, the path quantifiers are restricted to *fair paths*, that is, paths that intersect every fairness constraint infinitely often. A formula is said to hold in K if it is satisfied by every initial state of K. An ECTL formula is a CTL formula in which negation is only applied to the atomic propositions. An ACTL formula is the

*This work is supported in part by SRC contract 98-DJ-620 and NSF grant CR-99-71195.

DAC 2000, Los Angeles, California

$K, s_0 \models \varphi$	iff $\varphi \in L(s_0)$ for $\varphi \in A$
$K, s_0 \models \neg\varphi$	iff $\varphi \notin L(s_0)$ for $\varphi \in A$
$K, s_0 \models \varphi \vee \psi$	iff $K, s_0 \models \varphi$ or $K, s_0 \models \psi$
$K, s_0 \models \varphi \wedge \psi$	iff $K, s_0 \models \varphi$ and $K, s_0 \models \psi$
$K, s_0 \models \mathsf{EX}\varphi$	iff there exists a path $s_0, s_1, \ldots$ in K such that $K, s_1 \models \varphi$
$K, s_0 \models \mathsf{EG}\varphi$	iff there exists a path $s_0, s_1, \ldots$ in K such that for $i \geq 0$, $K, s_i \models \varphi$
$K, s_0 \models \mathsf{E}\varphi\,\mathsf{U}\,\psi$	iff there exists a path $s_0, s_1, \ldots$ in K such that there exists $i \geq 0$ for which $K, s_i \models \psi$, and for $0 \leq j < i$, $K, s_j \models \varphi$.

Figure 1: Semantics of CTL.

negation of an ECTL formula. A property that is neither ECTL nor ACTL is a *mixed property*.

Boolean operators other than $\wedge$, $\vee$, and $\neg$, and the operators EF, AX, AG, AF, and AU can be defined as abbreviations, e.g., $\mathsf{EF}\varphi = \mathsf{E}(\varphi \vee \neg\varphi)\,\mathsf{U}\,\varphi$, $\mathsf{AX}\varphi = \neg\mathsf{EX}\neg\varphi$, $\mathsf{AG}\varphi = \neg\mathsf{EF}\neg\varphi$, $\mathsf{AF}\varphi = \neg\mathsf{EG}\neg\varphi$, and $\mathsf{A}\varphi\,\mathsf{U}\,\psi = \neg(\mathsf{E}\neg\psi\,\mathsf{U}\,\neg(\varphi \vee \psi)) \wedge \neg\mathsf{EG}\neg\psi$. Clearly, the abbreviations should be expanded before checking whether a formula is an ECTL or ACTL formula. We refer the interested reader to [Eme90] for additional details and for a comparison of CTL to other temporal logics.

The model checking problem for CTL with fairness constraints can be translated into the computation of fixpoints of appropriate functionals [McM94]:

$$\mathsf{E}\varphi\,\mathsf{U}\,\psi = \mu Z.\,\psi \vee (\varphi \wedge \mathsf{EX}Z), \quad (1)$$

$$\mathsf{EG}\varphi = \nu Z.\,\varphi \wedge \mathsf{EX}Z, \quad (2)$$

$$\mathsf{E}_C\mathsf{G}\varphi = \nu Z.\,\varphi \wedge \mathsf{EX}\bigwedge_{c\in C}(\mathsf{E}Z\,\mathsf{U}\,(Z \wedge c))\ , \quad (3)$$

where C is a set of sets of states that must be traversed infinitely often by a fair path, and where with customary abuse of notation, we identify a formula and the set of states where it is satisfied. When identifying formulae with their satisfying set, we will sometimes annotate the formula with the transition relation it is evaluated under, to avoid confusion. The above formulation relies on *backward search*. Though most of the results presented in this paper apply to *forward* model checking [INH96] as well, we will limit our discussion to the traditional formulation. Most of the work in backward search is performed by the *preimage* computation, which computes all predecessors of a given set of states at each iteration of the fixpoint computation.

Note that EU, EG, and EX are monotonic both in their arguments and in the transition relation. Formally, let K be as before, and let $K' = \langle S, T', S_0, A, L\rangle$. If $T \subseteq T'$, $\varphi \subseteq \varphi'$, and $\psi \subseteq \psi'$, then $K, s \models \mathsf{EX}\varphi$ implies $K', s \models \mathsf{EX}\varphi'$, $K, s \models \mathsf{EG}\varphi$ implies $K', s \models \mathsf{EG}\varphi'$, and $K, s \models \mathsf{E}\varphi\,\mathsf{U}\,\psi$ implies $K', s \models \mathsf{E}\varphi'\,\mathsf{U}\,\psi'$. Since the operators $\wedge$ and $\vee$ are also monotonic, any ECTL formula φ that is true under transition relation T is also true under transition relation $T' \supseteq T$, and dually any ACTL formula φ that is false under T is also false under transition relation $T' \supseteq T$. Since negation is not monotonic, the corresponding statement can not be made for mixed properties.

3 Guided Search for CTL

3.1 Guided Search

CTL model checking algorithms compute a satisfying set for each formula and subformula. In subformulae that are expressed as fixpoints, the symbolic computations naturally result in a breadth-first exploration of the state graph.

A breadth-first search of the state space often suffers BDD size explosion. BDD sizes are dependent on the sets of states being explored (iterates of the fixpoint computation). Experimental evidence shows that the BDD sizes at the beginning and end of the computation are often much smaller than the intermediate sizes. (See Figure 3.) This suggests that the size explosion may be due in part to the breadth-first nature of the search; moreover it suggests that the BDD sizes may be controlled by modifying the search strategy [RS99a, CCLQ97, NIJ+97, RS99b].

In majority of model checking runs, large BDDs arise in *preimage* computation. The key factors in preimage computation are the states whose predecessors are being computed and the transition relation. By modifying either, the nature of the search may be modified. The flexibility available in modifying the transition relation is captured by the following definition and theorems.

Definition 1 *Let L be a finite lattice and let $T = \{\tau_1, \ldots, \tau_k\}$ be a finite set of monotonic functions $L \to L$. For $l_1, l_2 \in L$, let $l_1 \cup l_2$ ($l_1 \cap l_2$) be the least upper bound (greatest lower bound) of l_1 and l_2. For σ a finite sequence over T, let τ_σ be the function $L \to L$ obtained by composing all the functions in σ in the order specified by the sequence. We say that σ is* 0-closed *(*1-closed*) if, for $1 \leq i \leq k$, we have $\tau_i(\tau_\sigma(0)) = \tau_\sigma(0)$ ($\tau_i(\tau_\sigma(1)) = \tau_\sigma(1)$). An infinite sequence $\sigma = (\sigma_1, \ldots, \sigma_n, \ldots)$ over T is* fair *if, for $n > 0$ and for $0 < j \leq k$ there exists $i > n$ such that $\sigma_i = \tau_j$.*

Theorem 1 *Let $\gamma = \bigcup_{\tau\in T}\tau$ be the pointwise least upper bound of all the functions in T. Let $\hat{T} = \{\hat{\tau}_1, \ldots, \hat{\tau}_k\}$, with $\hat{\tau}_1 = \lambda x.x \cup \tau_i$. Let σ be a fair sequence over $\hat{T}$. Then σ has a finite prefix ρ that is 0-closed, and such that $\tau_\rho(0) = \mu x.\gamma$.*

Theorem 2 *Let $\beta = \bigcap_{\tau\in T}\tau$ be the pointwise greatest lower bound of all the functions in T. Let $\hat{T} = \{\hat{\tau}_1, \ldots, \hat{\tau}_k\}$, with $\hat{\tau}_1 = \lambda x.x \cap \tau_i$. Let σ be a fair sequence over $\hat{T}$. Then σ has a finite prefix ρ that is 1-closed, and such that $\tau_\rho(1) = \nu x.\beta$.*

Theorem 1 says that a set of underapproximations of the transition relation of a system can be used to compute the least fixpoint under the true transition relation, provided the approximations "add up" to the true relation. To ensure convergence to the fixpoint, one can apply the approximate transition relations until no one yields any new states. (This is, for instance, the approach of [CCLQ97, NIJ+97].) Alternatively, one can apply each approximation to convergence, as long as the last approximation is the original transition relation itself. (This is the approach of [RS99b, BRS99].) For greatest fixpoints we can dually use overapproximations of the transition relation. Notice that computing the least fixpoint under the original transition relation starting from the results obtained with the approximations may be dramatically more efficient than computing it from scratch.

Theorem 1 and Theorem 2 do not dictate (or even suggest) how the underapproximations should be derived from the transition relation of a system. In our application we mostly apply the method of [Rav99, BRS99]. This method tends to converge rapidly to the fixpoint without suffering a BDD size explosion, in spite of applying the original transition relation at the end of the computation. The approximations applied before restoring the original transition relation guide the state search in the fixpoint computation—hence the term *guided search*.

Symbolic guided search is accomplished with *hints*. Hints are assertions on the states or primary inputs of the system. Hints can be applied to restrict the behavior of the circuit by allowing only behavior that is consistent with the hint: transitions out of a state that violates the hint are disabled. This results in an underapproximation of the transition relation. The same hint can be applied to allow more behavior by allowing any transition from a state that violates

the hint, resulting in an overapproximation. This dual use of the hint parallels the dual characteristics of least and greatest fixpoints.

The strengths of guided search are threefold, addressing the most important bottlenecks in model checking.

1. Hints simplify the transition relation, making preimage computation more efficient.

2. Guided search is geared to exploring regular subsets of the state space, avoiding BDD blowup.

3. The results obtained in a simple system using one hint are employed towards the exact result, requiring less work in the more complicated exact system.

Hints are provided by the user, based on the knowledge of the design at hand, but require less effort on the part of the designer than detailed simulation stimuli, while retaining the complete coverage of model checking.

Hints may be property-dependent (when they direct the search towards states that will prove the property either true or false), system-dependent (when they exploit knowledge of the design to address the bottlenecks of computation), or both. By contrast, only property-dependent hints are applicable to non-symbolic guided search [YD98].

Hints are often easy to find; [RS99b] identifies types of hints that can be used to avoid difficult computations. These include choosing a simple mode of operation of a complex circuit (e.g., an ALU), or sequencing the use of registers in a bank to avoid BDD blowup due to symmetries. We will illustrate the use of hints for under- and overapproximation with an example.

3.1.1 Example

Gcd is a circuit implementing a variation of Euclid's algorithm [EucBC] for 8-bit numbers. We have two properties pertaining to the terminality of the algorithm. One is of the form $\mathsf{AG}(p \rightarrow \mathsf{EX}\,\mathsf{EX}\,\mathsf{EX}\,q)$, stating that a computation can always terminate within three clock ticks. The other, of the form $p \rightarrow \mathsf{AF}\,q$, states that a computation has to terminate eventually. The first property is false and the second is true, because although the algorithm may take more than three iterations, it does terminate.

Both properties are relatively hard to compute due to the wide data path. *Gcd* has two eight-bit numbers stored at any time. The two numbers are strongly interdependent, being the result of multiple iterations of Euclid's algorithm. The sets of states generated during an exploration of the states space are quite complex and hard to represent using BDDs.

Hence, we explore the circuit with a narrow datapath first, and then widen the datapath one bit at a time until we have included all eight bits. Thus, we fully explore the part of the state space that uses only the lower order bits, before we start exploration of the parts that also use the higher order bits. As evident from Table 1, this sequence of hints is quite successful for both properties. The exploration of the state space is very regular, and we avoid BDD blowup.

We discuss the effects of the hints on the least and greatest fixpoint computations in more detail in Section 3.2.1.

3.2 Under- and Overapproximations

When we compare CTL model checking to invariant checking [RS99b], or to the language emptiness check for *terminal* Büchi automata [BRS99], one finds three major differences: Both least and greatest fixpoints must be computed, nested fixpoints may be present, and state-space exploration is usually done backward instead of forward. In this section, we will discuss how to use guided search for both least and greatest fixpoints. In Section 3.3 we will discuss nested fixpoints. We will illustrate the effectiveness of guided search on backwards traversal in the examples.

in: Kripke structure (S, T, S_0, A, L),
hints $h_1, \ldots, h_n \subseteq S$, with $h_n = S$, and
sets of states p and q.
out: the set of states satisfying $[\mathsf{E}\,p\,\mathsf{U}\,q]_T$.

iterate = q;
for $h = h_1, \ldots, h_n$ **do**
  $T' = T \cap (h \times S)$;
  while iterate changes **do**
    iterate = iterate $\cup\,(p \cap [\mathsf{EX}\,(\text{iterate})]_{T'})$
  od
  // invariant: $[\mathsf{E}\,p\,\mathsf{U}\,q]_{T'} \subseteq$ iterate $\subseteq [\mathsf{E}\,p\,\mathsf{U}\,q]_T$
od
// invariant: iterate $= [\mathsf{E}\,p\,\mathsf{U}\,q]_T$
return iterate

Figure 2: Pseudo-code for computing $\mathsf{E}(p\,\mathsf{U}\,q)$ with (local) hints. Formulae are annotated with the transition relation they pertain to. Note that we do not require $h_i \subseteq h_{i+1}$.

The use of hints for least fixpoints is well known from [RS99b, BRS99]. One way to use hints for greatest fixpoints relies on the dual nature of least and greatest fixpoints.

For least fixpoints, we limit the states that are acquired at every iteration. Given a Kripke structure (S, T, S_0, A, L) and a hint h, we compute the underapproximation $T' = T \cap (h \times S)$ that consists of the transitions in T out of a state that satisfies h. Since T' contains no transitions from states that do not satisfy h, the iterate of Equation 1 does not acquire states that can only reach ψ via a transitions on which h is false. In particular, states that violate h are never visited. We show the pseudo-code for computation of a least fixpoint using hints in Figure 2.

For overapproximations, the iterate is monotonically shrinking. We hence limit the states that are *removed* at every iteration: To obtain an overapproximation from hint h, we change the transition relation of our system to $T' = T \cup ((S \setminus h) \times S)$, adding transitions from states violating h to all states. In particular, these states have a self loop. If we consider Equations 2 and 3, we see that this prevents states violating h from being removed from the iterate. Also, it prevents the removal of states that have a path within φ to a state that violates h.

For the least fixpoint, we prevent states from being added to the iterate if they satisfy $\neg\mathsf{E}(h\,\mathsf{U}\,h \wedge \psi)$. On the other hand, for greatest fixpoints, we prevent states from being removed from the iterate if they satisfy $\mathsf{E}(\varphi\,\mathsf{U}\,\varphi \wedge \neg h)$. Since the latter property is clearly weaker than the former (it only requires the existence of one path), hints for overapproximation may, if not chosen carefully, impede progress of the greatest fixpoint computation. In particular, hints that refer to primary inputs are of no use for overapproximation: They do not allow any states to be removed from the iterate.

3.2.1 Example

In *gcd*, we use the same hints for least and greatest fixpoints. For the first property we compute a least fixpoint. Here the algorithm approximates the transition relation by deleting any transitions from states with any of the higher order bits set. This restricts the states that are added to the iterate to those with the higher order bits set to zero. For the second property, we use an overapproximation that adds transitions from states with a higher order bit set to any state. This, dually, prevents such states from being *removed* from the iterate of the greatest fixpoint.

Note that in this case the hints combine quite well with backward exploration. in Figure 3, we see the BDD sizes during the computation of $p \rightarrow \mathsf{AF}\, q$. Note the seven humps in the computation with hints. Each one corresponds to the breadth-first exploration using one hint. The exploration using the first hint (computing the gcd of one-bit numbers) is too trivial to be apparent in the graph.

3.2.2 Incremental Fixpoint Update

An alternative to using underapproximations for least fixpoints and overapproximations for greatest fixpoints is to use incremental model checking. Incremental techniques [SS94, Swa96] address the problem of reevaluating a fixpoint after a change in the transition relation. In particular, suppose we wish to evaluate the greatest fixpoint $\mathsf{E}_C\mathsf{G}\, p$, in the Kripke structure (S, T, S_0, A, L), using underapproximations defined by hints h_1 and h_2. From these hints we obtain two transition relations $T_1 = T \cap (h_1 \times S)$, and $T_2 = T \cap (h_2 \times S)$. Note that in general, T_1 may contain edges that are not in T_2, and vice-versa.

First, we compute $S_1 = [\mathsf{E}_C\mathsf{G}\, p]_{T_1}$, the exact fixpoint using transition relation T_1. Then, to obtain the set $S_2 = [\mathsf{E}_C\mathsf{G}\, p]_{T_2}$, we could compute this value directly using T_2. This is wasteful, since it does not make use of the information obtained using hint h_1. Instead, we will compute an overapproximation S'' of S_2 with help of S_1, and use it as a starting point of our greatest fixpoint computation. Let D be the set of destination states of the transitions that are in T_2 but not in T_1, and let

$$\begin{aligned} F &= [\mathsf{E}_C\mathsf{G}\,\mathsf{true}]_{T_2} \\ S' &= [\mathsf{E}_C\mathsf{G}\, S_1]_{T_1 \cap T_2}, \\ S'' &= [\mathsf{E}\, p\ \mathsf{U}\ (p \wedge F \wedge (S' \vee D))]_{T_2}, \text{ and} \\ S''' &= [\mathsf{E}_C\mathsf{G}\, S'']_{T_2}. \end{aligned}$$

Here, F is the set of fair states of T_2. It is easily seen that $S' = [\mathsf{E}_C\mathsf{G}\, p]_{T1 \cap T2}$ ($T_1 \cap T_2$ has all the edges that appear in both T_1 and T_2), because S_1 is an overapproximation of this set. Also, $S' \subseteq S_2$. Using techniques from [SS94, Swa96] one can prove that S'' is an overapproximation of S_2. The basic underlying fact is that a cycle within p either exists in $T_1 \cap T_2$, or has an edge in it that is in T_2 but not in T_1. From $S'' \supseteq S_2$ it follows that $S''' = S_2$. A dual technique can be used for least fixpoints and overapproximations.

The set F may be expensive to compute, but we can use incremental techniques to compute this set, too, or we can replace it with an overapproximation that is more easily computed. One such overapproximation is true.

Note that if $S'' = p$, which can not always be prevented, we have wasted our effort, and our computation of S''' is equivalent to computing $[\mathsf{E}_C\mathsf{G}\, p]_{T_2}$ directly. Although results published in the incremental search literature are promising, we have not yet explored the practical benefits of this approach in the setting of guided search.

3.3 Nested Fixpoints

Guided search can be applied to nested fixpoint computations in several ways: Hints can be applied in sequence to each fixpoint in the formula until the fixpoint is computed (*locally*) or each hint can applied in succession to the formula as a whole (*globally*). The former approach computes the exact satisfying set of each fixpoint subformula, with the help of approximations of the fixpoints computed with each hint. Applying hints globally initially creates approximations of each subformula and with successive hints refines these satisfying sets. Whereas local hints can be applied to all CTL formulae, global hints are applicable to ECTL and ACTL properties only.

Consider the ECTL fragment and underapproximations for a discussion of global hints. We apply the hints in sequence, one at a time, to the original transition relation to create a subset. We then evaluate the entire formula under this transition relation, storing the satisfying sets of each subformula. The satisfying set of an ECTL subformula, computed with a subset of the transition relations is a subset of the exact satisfying set. (See Section 2.) Hence, if the satisfying set of the top formula contains all initial states, then the formula is true. If it does not, we apply the next hint, and reevaluate the formula using the new transition relation. The satisfying sets of the subformulae that have been computed during evaluation with the previous hint are used to start the next fixpoint computation.

The presence of negations on subformulae other than propositional formulae prevents the use of global hints, because we cannot mix under- and overapproximations. Hence, global hints cannot be applied to mixed properties. For such properties, we apply hints locally at every subformula. We thus compute the exact satisfying set for every subformula.

In our implementation, we allow for early termination of the fixpoint computations, as soon as it is clear that all initial states will satisfy the formula, or that some initial state will violate it. This criterion is only used for the outer fixpoint of a CTL formula. Guided search combines well with early termination because we may be able to decide the truth of a formula based on a simplified system, without having to compute the exact results of the fixpoints.

3.4 Analysis and Comparison to Prior Work

Some properties are inherently better suited to guided search than others. They make better use of the two important factors: Reuse of previous computations, and early termination.

Typically, least fixpoint computations can make good use of results computed under a previous approximation. A notable exception is an EU formula with an empty satisfying set. Evaluating such a formula in an underapproximated system will yield an empty set, causing no simplification in the next computation. This means that the evaluation under the hints is wasted effort. Such properties do occur in practice (See Section 4), often as invariants. They are a special case of the robust properties of [Eis99]. These properties are typically easy to prove without hints.

Using approximations of the transition relation to increase efficiency of model checking is a well-known technique. The authors of [CGL92, Kur94, LPJ+96, KMP98, CHM+96] have experimented with approximations as abstractions of the system. These approximations are constructed based on some information of the design. Other researchers [CCLQ97, NIJ+97, RS99b, PH97] have experimented with techniques based on controlling the BDD sizes. Guided search takes advantage of the knowledge of the system and applies hints that reduce the BDD sizes. The combined advantage offered may greatly benefit model checking in terms of effectiveness and applicability.

4 Experimental Results

All experiments were run on an IBM machine with a 400MHz Pentium II processor and 1GB of RAM. The memory use was limited to 750MB and runs were limited to one hour of CPU time. For every example, we performed model checking both with and without preceding reachability analysis. We report the best times for every instance.

Vsa is a scaled-down model of a DLX-style processor. It has four registers, of which register 0 always holds 0, and its datapath is five bits wide. The processor has the usual load, store, branch, arithmetic, and logical instructions. We check a true property of the form $\mathsf{EF}\, p$. The hints that we use are designed to explore the state space in a manner that is as regular as possible. First, we allow only loads to one register, going from register 1 to 3 in order. Then we do the branches, and the stores, and finally we use the original

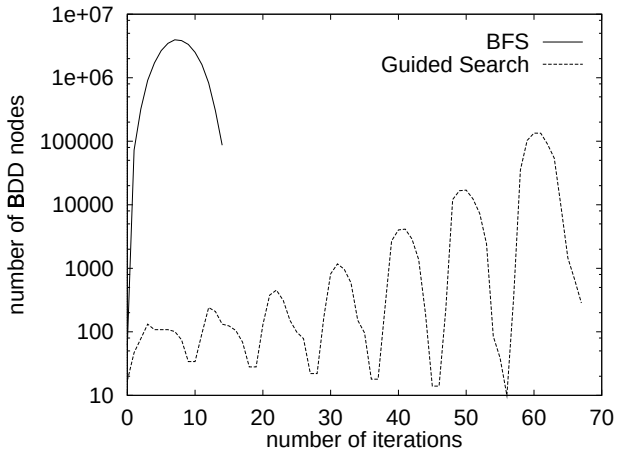

Figure 3: BDD size for model checking using breadth-first and guided search, on *Gcd* for the greatest-fixpoint property $p \rightarrow \mathsf{AF}\, q$. The sizes reported are the maximum size for intermediate products in each preimage computation.

transition relation. The hints sequence the use of the data registers to break symmetry between the registers, instead saturating them one by one [RS99b]. The hints successfully force the backward model checking algorithm to only consider states that could have been caused by certain instructions, preventing it from straying into parts of the state space that have complicated dependencies between various latches, such as the registers and the ALU input and output. This significantly reduces time, and especially memory consumption.

The second property we tried on this design was a mixed property of the form $\mathsf{AGEF}\, q$. The property passes, and although early termination does not help here, hints reduce time and memory use by about 25 and 30 percent respectively.

Soap is a model of a token-passing algorithm for distributed mutual exclusion [DK98], in which processors can be reset with an external input. The token is passed along a spanning tree of a network of processors. We use a property that passes, of the form $\mathsf{EF}\,(p \wedge \mathsf{EX}\,\mathsf{E}\, q\ \mathsf{U}\ r)$. The property is quite localized: It only pertains to two processors, and the ones in between them, that pass the token from one to the other. The sequence of hints we use turns on two processors at a time, leaving the rest disabled by keeping their reset input high. In this way, the algorithm explores a smaller instance of the *soap* model before moving on to a model with more processors. Early termination is crucial. It allows us to finish before ever turning on all processors.

Our third design is *Vsp*, a pipelined version of *Vsa*. For *Vsp* we have an invariant ($\mathsf{AG}\, p$), that holds. This is a good example of a property that does not allow for speedup using hints. In this design, the preimage of $\neg p$ under the exact transition relation is the empty set. Therefore, the preimage of $\neg p$ also equals the empty set under any subset of the relation. We have one hint here, plus the exact transition relation, and since the result of model checking using one hint cannot help the next one, we do the same work twice.

We have discussed *Gcd* in Section 3.1.1. Note that for the first, false property, early termination works well by finding the shortest possible counterexample.

Rcnum is an implementation of the "rollercoaster numbers". We are given the function

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even,} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases}.$$

Table 1: Performance of guided search.

experiment	procedure	time (s)	peak BDD nodes
Vsa	standard	140	5.5M
$\mathsf{EF}\, p$	early term.	79	5.3M
	hints	24	0.8M
	hints + e.t.	24	0.8M
Vsa	standard	58	4.7M
$\mathsf{AGEF}\, q$	early term.	58	4.7M
	hints	44	3.2M
	hints + e.t.	44	3.2M
Soap	standard	>3600	N/A
$\mathsf{EF}\,(p \wedge$	early term.	>3600	N/A
$\mathsf{EX}\,\mathsf{E}\, q\ \mathsf{U}\ r)$	hints	>3600	N/A
	hints + e.t.	38	4.1M
Vsp	standard	41	2.1M
$\mathsf{AG}\, p$	early term.	41	2.1M
	hints	72	2.5M
	hints + e.t.	72	2.5M
Gcd	standard	1600	20.5M
$\mathsf{AG}\,(p \rightarrow$	early term.	12	1.7M
$\mathsf{EX}\,\mathsf{EX}\,\mathsf{EX}\, q)$	hints	130	9.1M
	hints + e.t.	12	1.7M
Gcd	standard	1100	21.5M
$p \rightarrow \mathsf{AF}\, q$	early term.	1100	21.5M
	hints	22	2.0M
	hints + e.t.	22	2.0M
Rcnum	standard	2400	26.9M
$\mathsf{AF}\, p$	early term.	2400	26.9M
	hints	1000	15.0M
	hints + e.t.	1000	15.0M
Ethernet	standard	99	4.2M
$\mathsf{EFEG}\, p$	early term.	99	4.2M
	hints	88	6.2M
	hints + e.t.	88	6.2M

The question is whether for every natural number n there is an i such that $f^i(n) = 1$. This is an open problem. Our implementation uses 25 bit numbers, and changes f to return 0 when an overflow occurs. The true property that we check is that every computation eventually ends with the number 0 or 1, or in other words, that there is a path from any number to 0 or 1. Our hints are much like the hints for *gcd*. They keep the b higher order bits of the datapath set to 0 for decreasing b. Thus we first compute all numbers that have a path to 1 in a restricted system. Then we use these numbers to compute the numbers reaching 1 in a more relaxed system, making use of the information we have gained on the smaller datapath (any path will end with a tail of small numbers). Early termination does not help here, since every state is an initial state.

Ethernet models communication between a set of processors using the Ethernet protocol. This circuit has three processors. Each processor requests data non-deterministically and must receive data when a request is asserted. The transmitter module is responsible for transmitting data to the channel. A request is reset when the transmitter module signals success or failure. The property checks whether a processor can stay in a request asserted state forever. This property fails on this circuit, i.e., it is never possible to remain in a request state forever. The original model checking run takes 99 seconds. The property-dependent hint in this case asserts the conditions where the request must be reset, to force the property to fail. These condition are characterized by the transmitter module signaling success or failure. The hint is applied to obtain an overapproximation, so that when the processor is waiting for success/fail acknowledge from the transmitter transitions to any state is possi-

ble. The model checking run with hints is aided by the simplified transition relation and the resulting run takes 88 seconds.

5 Conclusions

We have presented an approach to efficient symbolic CTL model checking that relies on user-provided hints. The approach allows an exact evaluation of CTL formulas while avoiding BDD explosion and can be effectively combined with the detection of early termination conditions.

We have introduced the use of guided search for overapproximation, and shown the first results for backwards model checking and greatest fixpoint computations. We have discussed the possibility of using incremental model checking in combination with guided search, but we have not experimented with this option.

Guided search works well in avoiding BDD size explosion by simplifying the transition relation, and avoiding difficult-to-represent subsets of the state space. Employing hints provided by the user, the search concentrates on specific parts of the state space that are easy to represent and supply a good starting point for exploration of the entire state space.

Symbolic guided search speeds up CTL model checking and has been shown elsewhere to benefit LTL model checking as well as ω-regular language containment [BRS99]. Combining the two approaches yields a guided search algorithm for CTL* [EL87], which we plan to implement.

More work remains to be done in the integration of our approach with existing techniques for abstraction. In particular, we are interested in the combination of guided search with the iterative refinement schemes of [PH97, Jan99].

References

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, March 1999.

[BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*, pages 222–235. Springer-Verlag, Berlin, 1999. LNCS 1633.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[CBM89] 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, November 1989.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by constructions or approximation of fixpoints. In *Proceedings of the ACM Symposium on the Principles of Programming Languages*, pages 238–250, 1977.

[CCLQ97] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitioning 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.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings Workshop on Logics of Programs*, pages 52–71, Berlin, 1981. Springer-Verlag. LNCS 131.

[CGL92] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages*, January 1992.

[CHM+96] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. *IEEE Transactions on Computer-Aided Design*, 15(12):1465–1478, December 1996.

[DK98] J. Desel and E. Kindler. Proving correctness of distributed algorithms using high-level Petri nets: A case study. In *International Conference on Application of Concurrency to System Design*, Aizu, Japan, March 1998.

[Eis99] C. Eisner. Using symbolic model checking to verify te railway stations of Hoorn-Kersenboogerd and Heerhugowaard. In *Correct Hardware Design and Verification Methods (CHARME'99)*, pages 97–109, Bad Herrenalb, September 1999. Springer-Verlag. LNCS 1703.

[EL87] E. A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[Eme90] E. A. Emerson. Temporal and modal logic. In van Leeuwen [vL90], chapter 16, pages 995–1072.

[EucBC] Euclid. *Elements*, Book 7, ca. 300 B.C. English edition: T.L. Heath, ed., *The thirteen books of Euclid's Elements*, Dover Publications, New York, 1956.

[INH96] H. Iwashita, T. Nakata, and F. Hirose. CTL model checking based on forward state traversal. In *Proceedings of the International Conference on Computer-Aided Design*, pages 82–87, San Jose, CA, November 1996.

[Jan99] J.-Y. Jang. *Iterative Abstraction-based CTL Model Checking*. PhD thesis, University of Colorado, Department of Electrical and Computer Engineering, 1999.

[KMP98] M. Kaufmann, A. Martin, and C. Pixley. Design constraints in symbolic model checking. In A. J. Hu and M. Y. Vardi, editors, *Tenth Conference on Computer Aided Verification (CAV'98)*, pages 477–487. Springer-Verlag, Berlin, 1998. LNCS 1427.

[Kur94] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994.

[Kur97] R. P. Kurshan. Formal verification in a commercial setting. In *Proceedings of the Design Automation Conference*, pages 258–262, Anaheim, CA, June 1997.

[LPJ+96] W. Lee, A. Pardo, J. Jang, G. Hachtel, and F. Somenzi. Tearing based abstraction for CTL model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 76–81, San Jose, CA, November 1996.

[McM94] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[NIJ+97] A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Reachability analysis using partitioned ROBDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 388–393, November 1997.

[PH97] A. Pardo and G. D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In O. Grumberg, editor, *Ninth Conference on Computer Aided Verification (CAV'97)*, pages 12–23. Springer-Verlag, Berlin, 1997. LNCS 1254.

[Rav99] K. Ravi. *Adaptive Techniques to Improve State Space Search in Formal Verification*. PhD thesis, University of Colorado, Department of Electrical and Computer Engineering, 1999.

[RS99a] K. Ravi and F. Somenzi. Efficient fixpoint computation for invariant checking. In *Proceedings of the International Conference on Computer Design*, pages 467–474, Austin, TX, October 1999.

[RS99b] K. Ravi and F. Somenzi. Hints to accelerate symbolic traversal. In *Correct Hardware Design and Verification Methods (CHARME'99)*, pages 250–264, Berlin, September 1999. Springer-Verlag. LNCS 1703.

[SS94] O. V. Sokolsky and S. A. Smolka. Incremental model checking in the modal mu-calculus. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV '94)*, pages 351–363. Springer-Verlag, Berlin, 1994. LNCS 818.

[Swa96] G. Swamy. *Incremental Methods for Formal Verification and Logic Synthesis*. PhD thesis, University of California at Berkeley, 1996. UMI publication 9723211.

[vL90] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*. The MIT Press/Elsevier, Amsterdam, 1990.

[YD98] C. H. Yang and D. L. Dill. Validation with guided search of the state space. In *Proceedings of the Design Automation Conference*, pages 599–604, San Francisco, CA, June 1998.