Symbolic Model Checking without BDDs*

Armin Biere[1], Alessandro Cimatti[2], Edmund Clarke[1], and Yunshan Zhu[1]

[1] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A
{Armin.Biere,Edmund.Clarke,Yunshan.Zhu}@cs.cmu.edu
[2] Istituto per la Ricerca Scientifica e Tecnolgica (IRST)
via Sommarive 18, 38055 Povo (TN), Italy
cimatti@irst.itc.it

Abstract. Symbolic Model Checking [3, 14] has proven to be a powerful technique for the verification of reactive systems. BDDs [2] have traditionally been used as a symbolic representation of the system. In this paper we show how boolean decision procedures, like Stålmarck's Method [16] or the Davis & Putnam Procedure [7], can replace BDDs. This new technique avoids the space blow up of BDDs, generates counterexamples much faster, and sometimes speeds up the verification. In addition, it produces counterexamples of minimal length. We introduce a *bounded model checking* procedure for LTL which reduces model checking to propositional satisfiability. We show that bounded LTL model checking can be done without a tableau construction. We have implemented a model checker **BMC**, based on bounded model checking, and preliminary results are presented.

1 Introduction

Model checking [4] is a powerful technique for verifying reactive systems. Able to find subtle errors in real commercial designs, it is gaining wide industrial acceptance. Compared to other formal verification techniques (e.g. theorem proving) model checking is largely automatic.

In model checking, the specification is expressed in temporal logic and the system is modeled as a finite state machine. For realistic designs, the number of states of the system can be very large and the explicit traversal of the state space becomes infeasible. Symbolic model checking [3, 14], with boolean encoding of the finite state machine, can handle more than 10^{20} states. BDDs [2], a canonical form for boolean expressions, have traditionally been used as the underlying representation for symbolic model checkers [14]. Model checkers based on BDDs are usually able to handle systems with hundreds of state variables. However, for larger systems the BDDs generated during model checking become too large for currently available computers. In addition, selecting the right ordering of BDD variables is very important. The generation of a

* This research is sponsered by the National Science Foundation (NSF) under Grant No. CCR-9505472. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or the United States Government.

variable ordering that results in small BDDs is often time consuming or needs manual intervention. For many examples no space efficient variable ordering exists.

Propositional decision procedures (SAT) [7] also operate on boolean expressions but do not use canonical forms. They do not suffer from the potential space explosion of BDDs and can handle propositional satisfiability problems with thousands of variables. SAT based techniques have been successfully applied in various domains, such as hardware verification [17], modal logics [9], formal verification of railway control systems [1], and AI planning systems [11]. A number of efficient implementations are available. Some notable examples are the PROVE tool [1] based on Stålmarck's Method [16], and SATO [18] based on the Davis & Putnam Procedure [7].

In this paper we present a symbolic model checking technique based on SAT procedures. The basic idea is to consider counterexamples of a particular length k and generate a propositional formula that is satisfiable iff such a counterexample exists. In particular, we introduce the notion of *bounded model checking*, where the bound is the maximal length of a counterexample. We show that bounded model checking for linear temporal logic (LTL) can be reduced to propositional satisfiability in polynomial time. To prove the correctness and completeness of our technique, we establish a correspondence between bounded model checking and model checking in general. Unlike previous approaches to LTL model checking, our method does not require a tableau or automaton construction.

The main advantages of our technique are the following. First, bounded model checking finds counterexamples very fast. This is due to the depth first nature of SAT search procedures. Finding counterexamples is arguably the most important feature of model checking. Second, it finds counterexamples of minimal length. This feature helps the user to understand a counterexample more easily. Third, bounded model checking uses much less space than BDD based approaches. Finally, unlike BDD based approaches, bounded model checking does not need a manually selected variable order or time consuming dynamic reordering. Default splitting heuristics are usually sufficient.

To evaluate our ideas we have implemented a tool **BMC** based on bounded model checking. We give examples in which SAT based model checking significantly outperforms BDD based model checking. In some cases bounded model checking detects errors instantly, while the BDDs for the initial state cannot be built.

The paper is organized as follows. In the following section we explain the basic idea of bounded model checking with an example. In Section 3 we give the semantics for bounded model checking. Section 4 explains the translation of a bounded model checking problem into a propositional satisfiability problem. In Section 5 we discuss bounds on the length of counterexamples. In Section 6 our experimental results are presented, and Section 7 describes some directions for future research.

2 Example

Consider the following simple state machine M that consists of a three bit shift register x with the individual bits denoted by $x[0], x[1]$, and $x[2]$. The predicate $T(x, x')$ denotes the transition relation between current state values x and next state values x' and is equivalent to:

$$(x'[0] = x[1]) \wedge (x'[1] = x[2]) \wedge (x'[2] = 1)$$

In the initial state the content of the register x can be arbitrary. The predicate $I(x)$ that denotes the set of initial states is `true`.

This shift register is meant to be empty (all bits set to zero) after three consecutive shifts. But we introduced an error in the transition relation for the next state value of $x[2]$, where an incorrect value 1 is used instead of 0. Therefore, the property, that eventually the register will be empty (written as $x = 0$) after a sufficiently large number of steps is not valid. This property can be formulated as the LTL formula $\mathbf{F}(x = 0)$. We translate the "universal" model checking problem $\mathbf{AF}(x = 0)$ into the "existential" model checking problem $\mathbf{EG}(x \neq 0)$ by negating the formula. Then, we check if there is an execution sequence that fulfills $\mathbf{G}(x \neq 0)$. Instead of searching for an arbitrary path, we restrict ourselves to paths that have at most $k+1$ states, for instance we choose $k = 2$. Call the first three states of this path x_0, x_1 and x_2 and let x_0 be the initial state (see Figure 1). Since the initial content of x can be arbitrary, we do not have any restriction

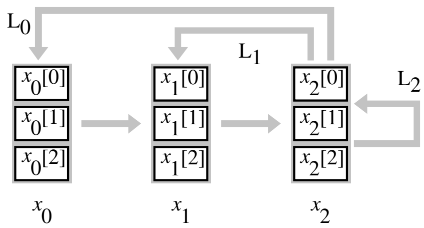

Fig. 1. Unrolling the transition relation twice and adding a back loop.

on x_0. We unroll the transition relation twice and derive the propositional formula f_m defined as $I(x_0) \wedge T(x_0, x_1) \wedge T(x_1, x_2)$. We expand the definition of T and I, and get the following formula.

$$\begin{array}{ll} (x_1[0] = x_0[1]) \;\wedge\; (x_1[1] = x_0[2]) \;\wedge\; (x_1[2] = 1) \;\wedge & \text{1st step} \\ (x_2[0] = x_1[1]) \;\wedge\; (x_2[1] = x_1[2]) \;\wedge\; (x_2[2] = 1) & \text{2nd step} \end{array}$$

Any path with three states that is a "witness" for $\mathbf{G}(x \neq 0)$ must contain a loop. Thus, we require that there is a transition from x_2 back to the initial state, to the second state, or to itself (see also Figure 1). We represent this transition as L_i defined as $T(x_2, x_i)$ which is equivalent to the following formula.

$$(x_i[0] = x_2[1]) \;\wedge\; (x_i[1] = x_2[2]) \;\wedge\; (x_i[2] = 1)$$

Finally, we have to make sure that this path will fulfill the constraints imposed by the formula $\mathbf{G}(x \neq 0)$. In this case the property S_i defined as $x_i \neq 0$ has to hold at each state. S_i is equivalent to the following formula.

$$(x_i[0] = 1) \vee (x_i[1] = 1) \vee (x_i[2] = 1)$$

Putting this all together we derive the following propositional formula.

$$f_M \wedge \bigvee_{i=0}^{2} L_i \wedge \bigwedge_{i=0}^{2} S_i \tag{1}$$

This formula is satisfiable iff there is a counterexample of length 2 for the original formula $\mathbf{F}(x = 0)$. In our example we find a satisfying assignment for (1) by setting $x_i[j] := 1$ for all $i, j = 0, 1, 2$.

3 Semantics

ACTL* is defined as the subset of formulas of CTL* [8] that are in negation normal form and contain only universal path quantifiers. A formula is in *negation normal form* (NNF) if negations only occur in front of atomic propositions. ECTL* is defined in the same way, but only existential path quantifiers are allowed. We consider the *next time* operator '**X**', the *eventuality* operator '**F**', the *globally* operator '**G**', and the *until* operator '**U**'. We assume that formulas are in NNF. We can always transform a formula in NNF without increasing its size by including the *release* operator '**R**' ($f\,\mathbf{R}\,g$ iff $\neg(\neg f\,\mathbf{U}\,\neg g)$). In an LTL formula no path quantifiers (**E** or **A**) are allowed. In this paper we concentrate on LTL model checking. Our technique can be extended to handle full ACTL* (resp. ECTL*).

Definition 1. *A* Kripke structure *is a tuple $M = (S, I, T, \ell)$ with a finite set of states S, the set of initial states $I \subseteq S$, a transition relation between states $T \subseteq S \times S$, and the labeling of the states $\ell\colon S \to \mathcal{P}(\mathcal{A})$ with* atomic propositions *$\mathcal{A}$.*

We use Kripke structures as models in order to give the semantics of the logic. For the rest of the paper we consider only Kripke structures for which we have a *boolean encoding*. We require that $S = \{0,1\}^n$, and that each state can be represented by a vector of state variables $s = (s(1), \ldots, s(n))$ where $s(i)$ for $i = 1, \ldots, n$ are propositional variables. We define propositional formulas $f_I(s)$, $f_T(s,t)$ and $f_p(s)$ as: $f_I(s)$ iff $s \in I$, $f_T(s,t)$ iff $(s,t) \in T$, and $f_p(s)$ iff $p \in \ell(s)$. For the rest of the paper we simply use $T(s,t)$ instead of $f_T(s,t)$ etc. In addition, we require that every state has a successor state. That is, for all $s \in S$ there is a $t \in S$ with $(s,t) \in T$. For $(s,t) \in T$ we also write $s \to t$. For an infinite sequence of states $\pi = (s_0, s_1, \ldots)$ we define $\pi(i) = s_i$ and $\pi^i = (s_i, s_{i+1}, \ldots)$ for $i \in \mathbb{N}$. An infinite sequence of states π is a *path* if $\pi(i) \to \pi(i+1)$ for all $i \in \mathbb{N}$.

Definition 2 (Semantics). *Let M be a Kripke structure, π be a path in M and f be an LTL formula. Then $\pi \models f$ (f is valid along π) is defined as follows.*

$$\begin{array}{lll@{\qquad}lll}
\pi \models p & \textit{iff} & p \in \ell(\pi(0)) & \pi \models \neg p & \textit{iff} & p \notin \ell(\pi(0)) \\
\pi \models f \wedge g & \textit{iff} & \pi \models f \textit{ and } \pi \models g & \pi \models f \vee g & \textit{iff} & \pi \models f \textit{ or } \pi \models g \\
\pi \models \mathbf{G} f & \textit{iff} & \forall i.\, \pi^i \models f & \pi \models \mathbf{F} f & \textit{iff} & \exists i.\, \pi^i \models f \\
\pi \models \mathbf{X} f & \textit{iff} & \pi^1 \models f & & & \\
\pi \models f\,\mathbf{U}\,g & \textit{iff} & \multicolumn{4}{l}{\exists i\,[\,\pi^i \models g \;\; \textit{and} \;\; \forall j,\, j < i.\, \pi^j \models f\,]} \\
\pi \models f\,\mathbf{R}\,g & \textit{iff} & \multicolumn{4}{l}{\forall i\,[\,\pi^i \models g \quad \textit{or} \quad \exists j,\, j < i.\, \pi^j \models f\,]}
\end{array}$$

Definition 3 (Validity). *An LTL formula f is* universally valid *in a Kripke structure M (in symbols $M \models \mathbf{A}f$) iff $\pi \models f$ for all paths π in M with $\pi(0) \in I$. An LTL formula f is* existentially valid *in a Kripke structure M (in symbols $M \models \mathbf{E}f$) iff there exists a path π in M with $\pi \models f$ and $\pi(0) \in I$.*

Determining whether an LTL formula f is existentially (resp. universally) valid in a given Kripke structure is called an *existential* (resp. *universal*) *model checking problem*.

In conformance to the semantics of CTL* [8], it is clear that an LTL formula f is universally valid in a Kripke structure M iff $\neg f$ is not existentially valid. In order to solve the universal model checking problem, we negate the formula and show that the existential model checking problem for the negated formula has no solution. Intuitively, we are trying to find a counterexample, and if we do not succeed then the formula is universally valid. Therefore, in the theory part of the paper we only consider the existential model checking problem.

The basic idea of *bounded model checking* is to consider only a *finite prefix* of a path that may be a solution to an existential model checking problem. We restrict the length of the prefix by a certain bound k. In practice we progressively increase the bound, looking for longer and longer possible counterexamples.

A crucial observation is that, though the prefix of a path is finite, it still might represent an infinite path if there is a *back loop* from the last state of the prefix to any of the previous states (see Figure 2(b)). If there is no such back loop (see Figure 2(a)), then the prefix does not say anything about the infinite behavior of the path. For instance, only a prefix with a back loop can represent a witness for $\mathbf{G}p$. Even if p holds along all the states from s_0 to s_k, but there is no back loop from s_k to a previous state, then we cannot conclude that we have found a witness for $\mathbf{G}p$, since p might not hold at s_{k+1}.

(a) no loop   (b) (k,l)-loop

Fig. 2. The two cases for a *bounded* path.

Definition 4. *For $l \leq k$ we call a path π a (k,l)*-loop *if $\pi(k) \rightarrow \pi(l)$ and $\pi = u \cdot v^\omega$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$. We call π simply a k*-loop *if there is an $l \in \mathbb{N}$ with $l \leq k$ for which π is a (k,l)-loop.*

We give a *bounded* semantics that is an approximation to the unbounded semantics of Definition 2. It allows us to define the bounded model checking problem and in the next section we will give a translation of a bounded model checking problem into a satisfiability problem.

In the bounded semantics we only consider a finite prefix of a path. In particular, we only use the first $k+1$ states $(s_0, \ldots, s_k)$ of a path to determine the validity of a formula along that path. If a path is a k-loop then we simply maintain the original LTL semantics, since all the information about this (infinite) path is contained in the prefix of length k.

Definition 5 (Bounded Semantics for a Loop). *Let $k \in \mathbb{N}$ and π be a k-loop. Then an LTL formula f is* valid along the path π with bound k *(in symbols $\pi \models_k f$) iff $\pi \models f$.*

Assume that π is not a k-loop. Then the formula $f := \mathbf{F}p$ is valid along π in the unbounded semantics if we can find an index $i \in \mathbb{N}$ such that p is valid along the suffix π^i of π. In the bounded semantics the $(k+1)$-th state $\pi(k)$ does not have a successor. Therefore, we cannot define the bounded semantics recursively over *suffixes* (e.g. π^i) of π. We keep the original π instead but add a parameter i in the definition of the bounded semantics and use the notation $\models_k^i$. The parameter i is the current position in the prefix of π. In Lemma 7 we will show that $\pi \models_k^i f$ implies $\pi^i \models f$.

Definition 6 (Bounded Semantics without a Loop). *Let $k \in \mathbb{N}$, and let π be a path that is* not *a k-loop. Then an LTL formula f is* valid along π with bound k *(in symbols $\pi \models_k f$) iff $\pi \models_k^0 f$ where*

$$
\begin{array}{llllll}
\pi \models_k^i p & \textit{iff} & p \in \ell(\pi(i)) & \pi \models_k^i \neg p & \textit{iff} & p \notin \ell(\pi(i)) \\
\pi \models_k^i f \wedge g & \textit{iff} & \pi \models_k^i f \textit{ and } \pi \models_k^i g & \pi \models_k^i f \vee g & \textit{iff} & \pi \models_k^i f \textit{ or } \pi \models_k^i g \\
\pi \models_k^i \mathbf{G}f & \multicolumn{2}{l}{\textit{is always false}} & \pi \models_k^i \mathbf{F}f & \textit{iff} & \exists j,\, i \leq j \leq k.\, \pi \models_k^j f \\
\pi \models_k^i \mathbf{X}f & \textit{iff} & i < k \textit{ and } \pi \models_k^{i+1} f & & & \\
\pi \models_k^i f\,\mathbf{U}\,g & \textit{iff} & \multicolumn{4}{l}{\exists j,\, i \leq j \leq k\,[\,\pi \models_k^j g \;\; \textit{and} \;\; \forall n,\, i \leq n < j.\, \pi \models_k^n f\,]} \\
\pi \models_k^i f\,\mathbf{R}\,g & \textit{iff} & \multicolumn{4}{l}{\exists j,\, i \leq j \leq k\,[\,\pi \models_k^j f \;\; \textit{and} \;\; \forall n,\, i \leq n \leq j.\, \pi \models_k^n g\,]}
\end{array}
$$

Note that if π is not a k-loop, then we say that $\mathbf{G}f$ is not valid along π in the bounded semantics with bound k since f might not hold along π^{k+1}. Similarly, the case for $f\,\mathbf{R}\,g$ where g always holds and f is never fulfilled has to be excluded. These constraints imply that for the bounded semantics the duality of $\mathbf{G}$ and $\mathbf{F}$ ($\neg\mathbf{F}f \equiv \mathbf{G}\neg f$) and the duality of $\mathbf{R}$ and $\mathbf{U}$ ($\neg(f\,\mathbf{U}\,g) \equiv (\neg f)\,\mathbf{R}\,(\neg g)$) no longer hold.

The existential and universal bounded model checking problems are defined in the same manner as in Definition 3. Now we describe how the existential model checking problem ($M \models \mathbf{E}f$) can be reduced to a *bounded* existential model checking problem ($M \models_k \mathbf{E}f$).

Lemma 7. *Let h be an LTL formula and π a path, then $\pi \models_k h \;\Rightarrow\; \pi \models h$*

Proof. If π is a k-loop then the conclusion follows by definition. In the other case we assume that π is not a loop. Then we prove by induction over the structure of f and $i \leq k$ the stronger property $\pi \models_k^i h \;\Rightarrow\; \pi^i \models h$. We only consider the most complicated case $h = f\,\mathbf{R}\,g$.

$$
\begin{aligned}
\pi \models_k^i f\,\mathbf{R}\,g \;&\Leftrightarrow\; \exists j,\, i \leq j \leq k\,[\,\pi \models_k^j f \;\text{ and }\; \forall n,\, i \leq n \leq j.\, \pi \models_k^n g\,] \\
&\Rightarrow\; \exists j,\, i \leq j \leq k\,[\,\pi^j \models f \;\text{ and }\; \forall n,\, i \leq n \leq j.\, \pi^n \models g\,] \\
&\Rightarrow\; \exists j,\, i \leq j\,[\,\pi^j \models f \;\text{ and }\; \forall n,\, i \leq n \leq j.\, \pi^n \models g\,]
\end{aligned}
$$

Let $j' = j - i$ and $n' = n - i$

$$
\begin{aligned}
&\Rightarrow \quad \exists j' \, [\, \pi^{i+j'} \models f \;\text{ and }\; \forall n',\, n' \leq j'.\, \pi^{i+n'} \models g \,] \\
&\Rightarrow \quad \exists j \, [\, (\pi^i)^j \models f \;\text{ and }\; \forall n,\, n \leq j.\, (\pi^i)^n \models g \,] \\
&\Rightarrow \quad \forall n \, [\, (\pi^i)^n \models g \;\text{ or }\; \exists j,\, j < n.\, (\pi^i)^j \models f \,] \\
&\Rightarrow \quad \pi^i \models f \,\mathbf{R}\, g
\end{aligned}
$$

In the next-to-last step we used the following fact:

$$\exists m \, [\, \pi^m \models f \text{ and } \forall l,\, l \leq m.\, \pi^l \models g \,] \quad \Rightarrow \quad \forall n \, [\, \pi^n \models g \text{ or } \exists j,\, j < n.\, \pi^j \models f \,]$$

Assume that m is the smallest number such that $\pi^m \models f$ and $\pi^l \models g$ for all l with $l \leq m$. In the first case we consider $n > m$. Based on the assumption, there exists $j < n$ such that $\pi^j \models f$ (choose $j = m$). The second case is $n \leq m$. Because $\pi^l \models g$ for all $l \leq m$ we have $\pi^n \models g$ for all $n \leq m$. Thus, for all n we have proven that the disjunction on the right hand side is fulfilled. □

Lemma 8. *Let f be an LTL formula f and M a Kripke structure. If $M \models \mathbf{E}f$ then there exists $k \in \mathbb{N}$ with $M \models_k \mathbf{E}f$*

Proof. In [3, 5, 12] it is shown that an existential model checking problem for an LTL formula f can be reduced to FairCTL model checking of the formula $\mathbf{EG}\texttt{true}$ in a certain product Kripke structure. This Kripke structure is the product of the original Kripke structure and a "tableau" that is exponential in the size of the formula f in the worst case. If the LTL formula f is existentially valid in M then there exists a path in the product structure that starts with an initial state and ends with a cycle in the strongly connected component of fair states. This path can be chosen to be a k-loop with k bounded by $|S| \cdot 2^{|f|}$ which is the size of the product structure. If we project this path onto its first component, the original Kripke structure, then we get a path π that is a k-loop and in addition fulfills $\pi \models f$. By definition of the bounded semantics this also implies $\pi \models_k f$. □

The main theorem of this section states that, if we take all possible bounds into account, then the bounded and unbounded semantics are equivalent.

Theorem 9. *Let f be an LTL formula, M a Kripke structure. Then $M \models \mathbf{E}f$ iff there exists $k \in \mathbb{N}$ with $M \models_k \mathbf{E}f$.*

4 Translation

In the previous section, we defined the semantics for bounded model checking. We now reduce bounded model checking to propositional satisfiability. This reduction enables us to use efficient propositional decision procedures to perform model checking.

Given a Kripke structure M, an LTL formula f and a bound k, we will construct a propositional formula $[\![\, M, f \,]\!]_k$. The variables $s_0, \ldots, s_k$ in $[\![\, M, f \,]\!]_k$ denote a finite sequence of states on a path π. Each s_i is a vector of state variables. The formula $[\![\, M, f \,]\!]_k$

essentially represents constraints on $s_0, \ldots, s_k$ such that $[\![\, M, f \,]\!]_k$ is satisfiable iff f is valid along π.

The size of $[\![\, M, f \,]\!]_k$ is polynomial in the size of f if common subformulas are shared (as in our tool **BMC**). It is linear in k and in the size of the propositional formulas for T, I and the $p \in \mathcal{A}$. Thus, existential bounded model checking can be reduced in polynomial time to propositional satisfiability.

To construct $[\![\, M, f \,]\!]_k$, we first define a propositional formula $[\![\, M \,]\!]_k$ that constrains $s_0, \ldots, s_k$ to be on a valid path π in M. Second, we give the translation of an LTL formula f to a propositional formula that constrains π to satisfy f.

Definition 10 (Unfolding the Transition Relation). *For a Kripke structure M, $k \in \mathbb{N}$*

$$[\![\, M \,]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

Depending on whether a path is a k-loop or not (see Figure 2), we have two different translations of the temporal formula f. In Definition 11 we describe the translation if the path is not a loop ("$[\![\, \cdot \,]\!]_k^i$"). The more technical translation where the path is a loop ("${}_l[\![\, \cdot \,]\!]_k^i$") is given in Definition 13.

Consider the formula $h := p \,\mathbf{U}\, q$ and a path π that is not a k-loop for a given $k \in \mathbb{N}$ (see Figure 2(a)). Starting at π^i for $i \in \mathbb{N}$ with $i \le k$ the formula h is valid along π^i with respect to the bounded semantics iff there is a position j with $i \le j \le k$ and q holds at $\pi(j)$. In addition, for all states $\pi(n)$ with $n \in \mathbb{N}$ starting at $\pi(i)$ up to $\pi(j-1)$ the proposition p has to be fulfilled. Therefore the translation is simply a disjunction over all possible positions j at which q eventually might hold. For each of these positions a conjunction is added that ensures that p holds along the path from $\pi(i)$ to $\pi(j-1)$. Similar reasoning leads to the translation of the other temporal operators.

The translation "$[\![\, \cdot \,]\!]_k^i$" maps an LTL formula into a propositional formula. The parameter k is the length of the prefix of the path that we consider and i is the current position in this prefix (see Figure 2(a)). When we recursively process subformulas, i changes but k stays the same. Note that we define the translation of any formula $\mathbf{G}f$ as false. This translation is consistent with the bounded semantics.

Definition 11 (Translation of an LTL Formula without a Loop). *For an LTL formula f and $k, i \in \mathbb{N}$, with $i \le k$*

$$\begin{array}{rcl@{\qquad}rcl}
[\![\, p \,]\!]_k^i & := & p(s_i) & [\![\, \neg p \,]\!]_k^i & := & \neg p(s_i) \\
[\![\, f \wedge g \,]\!]_k^i & := & [\![\, f \,]\!]_k^i \wedge [\![\, g \,]\!]_k^i & [\![\, f \vee g \,]\!]_k^i & := & [\![\, f \,]\!]_k^i \vee [\![\, g \,]\!]_k^i \\
[\![\, \mathbf{G}f \,]\!]_k^i & := & \mathsf{false} & [\![\, \mathbf{F}f \,]\!]_k^i & := & \bigvee_{j=i}^{k} [\![\, f \,]\!]_k^j \\
[\![\, \mathbf{X}f \,]\!]_k^i & := & \multicolumn{4}{l}{\textit{if } i < k \textit{ then } [\![\, f \,]\!]_k^{i+1} \textit{ else } \mathsf{false}} \\
[\![\, f \,\mathbf{U}\, g \,]\!]_k^i & := & \multicolumn{4}{l}{\bigvee_{j=i}^{k} \left([\![\, g \,]\!]_k^j \wedge \bigwedge_{n=i}^{j-1} [\![\, f \,]\!]_k^n \right)} \\
[\![\, f \,\mathbf{R}\, g \,]\!]_k^i & := & \multicolumn{4}{l}{\bigvee_{j=i}^{k} \left([\![\, f \,]\!]_k^j \wedge \bigwedge_{n=i}^{j} [\![\, g \,]\!]_k^n \right)}
\end{array}$$

Now we consider the case where the path is a k-loop. The translation "${}_l[\![\cdot]\!]_k^i$" of an LTL formula depends on the current position i and on the length of the prefix k. It also depends on the position where the loop starts (see Figure 2(b)). This position is denoted by l for *l*oop.

Definition 12 (Successor in a Loop). *Let $k, l, i \in \mathbb{N}$, with $l, i \leq k$. Define the successor* $\operatorname{succ}(i)$ *of i in a (k,l)-loop as* $\operatorname{succ}(i) := i+1$ *for $i < k$ and* $\operatorname{succ}(i) := l$ *for $i = k$.*

Definition 13 (Translation of an LTL Formula for a Loop). *Let f be an LTL formula, $k, l, i \in \mathbb{N}$, with $l, i \leq k$.*

$$
\begin{array}{rclrcl}
{}_l[\![p]\!]_k^i & := & p(s_i) & {}_l[\![\neg p]\!]_k^i & := & \neg p(s_i) \\
{}_l[\![f \wedge g]\!]_k^i & := & {}_l[\![f]\!]_k^i \wedge {}_l[\![g]\!]_k^i & {}_l[\![f \vee g]\!]_k^i & := & {}_l[\![f]\!]_k^i \vee {}_l[\![g]\!]_k^i \\
{}_l[\![\mathbf{G} f]\!]_k^i & := & \bigwedge_{j=\min(i,l)}^k {}_l[\![f]\!]_k^j & {}_l[\![\mathbf{F} f]\!]_k^i & := & \bigvee_{j=\min(i,l)}^k {}_l[\![f]\!]_k^j \\
{}_l[\![\mathbf{X} f]\!]_k^i & := & {}_l[\![f]\!]_k^{\operatorname{succ}(i)} & & & \\
{}_l[\![f\, \mathbf{U}\, g]\!]_k^i & := & \multicolumn{4}{l}{\bigvee_{j=i}^k \left({}_l[\![g]\!]_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l[\![f]\!]_k^n \right) \vee} \\
 & & \multicolumn{4}{l}{\bigvee_{j=l}^{i-1} \left({}_l[\![g]\!]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[\![f]\!]_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l[\![f]\!]_k^n \right)} \\
{}_l[\![f\, \mathbf{R}\, g]\!]_k^i & := & \multicolumn{4}{l}{\bigwedge_{j=\min(i,l)}^k {}_l[\![g]\!]_k^j \vee} \\
 & & \multicolumn{4}{l}{\bigvee_{j=i}^k \left({}_l[\![f]\!]_k^j \wedge \bigwedge_{n=i}^{j} {}_l[\![g]\!]_k^n \right) \vee} \\
 & & \multicolumn{4}{l}{\bigvee_{j=l}^{i-1} \left({}_l[\![f]\!]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[\![g]\!]_k^n \wedge \bigwedge_{n=l}^{j} {}_l[\![g]\!]_k^n \right)}
\end{array}
$$

The translation of the formula depends on the shape of the path (whether it is a loop or not). We now define a loop condition to distinguish these cases.

Definition 14 (Loop Condition). *For $k, l \in \mathbb{N}$, let* ${}_l\mathrm{L}_k := T(s_k, s_l)$, $\mathrm{L}_k := \bigvee_{l=0}^k {}_l\mathrm{L}_k$

Definition 15 (General Translation). *Let f be an LTL formula, M a Kripke structure and $k \in \mathbb{N}$*

$$
[\![M, f]\!]_k := [\![M]\!]_k \wedge \left(\left(\neg \mathrm{L}_k \wedge [\![f]\!]_k^0 \right) \vee \bigvee_{l=0}^{k} \left({}_l\mathrm{L}_k \wedge {}_l[\![f]\!]_k^0 \right) \right)
$$

The left side of the disjunction is the case where there is no back loop and the translation without a loop is used. On the right side all possible starts l of a loop are tried and the translation for a (k,l)-loop is conjuncted with the corresponding ${}_lL_k$ loop condition.

Theorem 16. *$[\![M, f]\!]_k$ is satisfiable iff $M \models_k \mathbf{E} f$.*

Corollary 17. *$M \models \mathbf{A} \neg f$ iff $[\![M, f]\!]_k$ is unsatisfiable for all $k \in \mathbb{N}$.*

5 Determining the bound

In Section 3 we have shown that the unbounded semantics is equivalent to the bounded semantics if we consider all possible bounds. This equivalence leads to a straightforward LTL model checking procedure. To check whether $M \models \mathbf{E}f$, the procedure checks $M \models_k \mathbf{E}f$ for $k = 0, 1, 2, \ldots$. If $M \models_k \mathbf{E}f$, then the procedure proves that $M \models \mathbf{E}f$ and produces a witness of length k. If $M \not\models \mathbf{E}f$, we have to increment the value of k indefinitely, and the procedure does not terminate. In this section we establish several bounds on k. If $M \not\models_k \mathbf{E}f$ for all k within the bound, we conclude that $M \not\models \mathbf{E}f$.

5.1 ECTL

ECTL is a subset of ECTL* where each temporal operator is preceded by one existential path quantifier. We have extended bounded model checking to handle ECTL formulas. Semantics and translation for ECTL formulas can be found in the full version of this paper. In general, better bounds can be derived for ECTL formulas than for LTL formulas. The intersection of the two sets of formulas includes many temporal properties of practical interest (e.g. $\mathbf{EF}p$ and $\mathbf{EG}p$). Therefore, we include the discussion of bounds for ECTL formulas in this section.

Theorem 18. *Given an ECTL formula f and a Kripke structure M. Let $|M|$ be the number of states in M, then $M \models \mathbf{E}f$ iff there exists $k \leq |M|$ with $M \models_k \mathbf{E}f$.*

In symbolic model checking, the number of states in a Kripke structure is bounded by 2^n, where n is the number of boolean variables to encode the Kripke structure. Typical model checking problems involve Kripke structures with tens or hundreds of boolean variables. The bound given in Theorem 18 is often too large for practical problems.

Definition 19 (Diameter). *Given a Kripke structure M, the* diameter *of M is the minimal number $d \in \mathbb{N}$ with the following property. For every sequence of states $s_0, \ldots, s_{d+1}$ with $(s_i, s_{i+1}) \in T$ for $i \leq d$, there exists a sequence of states $t_0, \ldots, t_l$ where $l \leq d$ such that $t_0 = s_0$, $t_l = s_{d+1}$ and $(t_j, t_{j+1}) \in T$ for $j < l$. In other words, if a state v is reachable from a state u, then v is reachable from u via a path of length d or less.*

Theorem 20. *Given an ECTL formula $f := \mathbf{EF}p$ and a Kripke structure M with diameter d, $M \models \mathbf{EF}p$ iff there exists $k \leq d$ with $M \models_k \mathbf{EF}p$.*

Theorem 21. *Given a Kripke structure M, its diameter d is the minimal number that satisfies the following formula.*

$$\forall s_0, \ldots, s_{d+1}.\ \exists t_0, \ldots, t_d.\ \bigwedge_{i=0}^{d} T(s_i, s_{i+1}) \rightarrow (t_0 = s_0 \wedge \bigwedge_{i=0}^{d-1} T(t_i, t_{i+1}) \wedge \bigvee_{i=0}^{d} t_i = s_{d+1})$$

For a Kripke structure with explicit state representation, well-known graph algorithms can be used to determine its diameter. For a Kripke structure M with a boolean encoding, one may verify that d is indeed a diameter of M by evaluating a quantified boolean formula (QBF), shown in Theorem 21. We conjecture that a quantified boolean formula is necessary to express the property that d is the diameter of M. Unfortunately, we do not know of an efficient decision procedure for QBF.

Definition 22 (Recurrence Diameter). *Given a Kripke structure M, its* recurrence diameter *is the minimal number $d \in \mathbb{N}$ with the following property. For every sequence of states $s_0, \ldots, s_{d+1}$ with $(s_i, s_{i+1}) \in T$ for $i \leq d$, there exists $j \leq d$ such that $s_{d+1} = s_j$.*

Theorem 23. *Given an ECTL formula f and a Kripke structure M with recurrence diameter d, $M \models \mathbf{E}f$ iff there exists $k \leq d$ with $M \models_k \mathbf{E}f$.*

Theorem 24. *Given any Kripke structure M, its recurrence diameter d is the minimal number that satisfies the following formula*

$$\forall s_0, \ldots, s_{d+1}. \bigwedge_{i=0}^{d} T(s_i, s_{i+1}) \rightarrow \bigvee_{i=0}^{d} s_i = s_{d+1}$$

The recurrence diameter in Definition 22 is a bound on k for bounded model checking that is applicable for all ECTL formulas. The property of a recurrence diameter can be expressed as a propositional formula as shown in Theorem 24. We may use a propositional decision procedure to determine whether a number d is the recurrence diameter of a Kripke structure. The bound based on recurrence diameter is not as tight as that based on the diameter. For example, in a fully connected Kripke structure, the graph diameter is 1 while the recurrence diameter equals the number of states.

5.2 LTL

LTL model checking is known to be PSPACE-complete [15]. In section 4, we reduced bounded LTL model checking to propositional satisfiability and thus showed that it is in NP. Therefore, a polynomial bound on k with respect to the size of M and f for which $M \models_k \mathbf{E}f \Leftrightarrow M \models \mathbf{E}f$ is unlikely to be found. Otherwise, there would be a polynomial reduction of LTL model checking problems to propositional satisfiability and thus PSPACE = NP.

Theorem 25. *Given an LTL formula f and a Kripke structure M, let $|M|$ be the number of states in M, then $M \models \mathbf{E}f$ iff there exists $k \leq |M| \times 2^{|f|}$ with $M \models_k \mathbf{E}f$.*

For the subset of LTL formulas that involves only temporal operators **F** and **G**, LTL model checking is NP-complete [15]. For this subset of LTL formulas, it can be shown that there exists a bound on k linear in the number of states and the size of the formula.

Definition 26 (Loop Diameter). *We say a Kripke structure M is* lasso shaped *if every path p starting from an initial state is of the form $u_p v_p^{\omega}$, where u_p and v_p are finite sequences of length less or equal to u and v, respectively. We define the* loop diameter *of M as (u, v).*

Theorem 27. *Given an LTL formula f and a lasso-shaped Kripke structure M, let the loop diameter of M be (u, v), then $M \models \mathbf{E}f$ iff there exists $k \leq u + v$ with $M \models_k \mathbf{E}f$.*

Theorem 27 shows that for a restricted class of Kripke structures, small bounds on k exist. In particular, if a Kripke structure is lasso shaped, k is bounded by $u + v$, where (u, v) is the loop diameter of M.

6 Experimental Results

We have implemented a model checker **BMC** based on bounded model checking. Its input language is a subset of the SMV language [14]. It outputs a SMV program or a propositional formula. For the propositional output mode, two different formats are supported. The first format is the DIMACS format [10] for satisfiability problems. The SATO tool [18] is a very efficient implementation of the Davis & Putnam Procedure [7] and it uses the DIMACS format. We also support the input format of the PROVE Tool [1] which is based on Stålmarck's Method [16].

As benchmarks we chose examples where BDDs are known to behave badly. First we investigated a sequential multiplier, the sequential shift and add multiplier of [6]. We formulated as *model checking* problem the following property: when the sequential multiplier is finished its output is the same as the output of a combinational multiplier (the C6288 circuit from the ISCAS'85 benchmarks) applied to the same input words. These multipliers are 16x16 bit multipliers but we only allowed 16 output bits as in [6] together with an overflow bit. We proved the property for each output bit individually and the results are shown in Table 1. For SATO we conducted two experiments to study the effect of the '-g' parameter that controls the maximal size of cached clauses. We picked a very small value ('-g 5') and a very large value ('-g 50'). Note that the overflow bit depends on all the bits of the sequential multiplier and occurs in the specification. Thus, cone of influence reduction could not remove anything.

	SMV_1		SMV_2		SATO -g5		SATO -g50		PROVE	
bit	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB
0	919	13	25	79	0	0	0	1	0	1
1	1978	13	25	79	0	0	0	1	0	1
2	2916	13	26	80	0	0	0	2	0	1
3	4744	13	27	82	0	0	0	3	1	2
4	6580	15	33	92	2	0	3	4	1	2
5	10803	25	67	102	12	0	36	7	1	2
6	43983	73	258	172	55	0	208	10	2	2
7	>17h		1741	492	209	0	642	13	7	3
8				>1GB	473	0	1198	16	29	3
9					856	1	2413	20	58	3
10					1837	1	2055	20	91	3
11					2367	1	1667	19	125	3
12					3830	1	976	17	156	4
13					5128	1	4363	25	186	4
14					4752	1	2170	23	226	4
15					4449	1	6847	31	183	5
sum	71923		2202		23970		22578		1066	

Table 1. 16x16 bit sequential shift and add multiplier with overflow flag and 16 output bits (sec = seconds, MB = Mega Byte).

In the column SMV_1 of Table 1 the official version of the CMU model checker SMV was used. SMV_2 is a version by Bwolen Yang from CMU with improved support for conjunctive partitioning. We used a manually chosen variable ordering where the bits of registers are interleaved. Dynamic reordering failed to find a considerably better ordering in a reasonable amount of time.

We used a barrel shifter as another example. It rotates the contents of a register file b with each step by one position. The model also contains another register file r that is related to b in the following way. If a register in r and one in b have the same contents then their neighbors also have the same contents. This property holds in the initial state of the model, and we proved that it is valid in all successor states. The results of this experiment can be found in Table 2. The width of the registers is chosen to be $\lceil \log_2 |r| \rceil$ where $|r|$ is the number of registers in the register file r. In this case we were also able to prove the recurrence diameter (see Definition 22) to be $|r|$. This took only very little time compared to the total verification time and is shown in the column "diameter".

In [13] an asynchronous circuit for distributed mutual exclusion is described. It consists of n cells for n users that want to have exclusive access to a shared resource. We proved the liveness property that a request for using the resource will eventually be acknowledged. This liveness property is only true if each asynchronous gate does not delay execution indefinitely. We model this assumption by a fairness constraint for each individual gate. Each cell has exactly 18 gates and therefore the model has $n \cdot 18$ fairness constraints where n is the number of cells. Since we do not have a bound for the maximal length of a counterexample for the verification of this circuit we could not verify the liveness property completely. We only showed that there are no counterexamples of particular length k. To illustrate the performance of bounded model checking we have chosen $k = 5, 10$. The results can be found in Table 3.

We repeated the experiment with a buggy design. For the liveness property we simply removed several fairness constraints. Both PROVE and SATO generate a counterexample (a 2-loop) instantly (see Table 4).

7 Conclusion

This work is the first step in applying SAT procedures to symbolic model checking. We believe that our technique has the potential to handle much larger designs than what is currently possible. Towards this goal, we propose several promising directions of research. We would like to investigate how to use domain knowledge to guide the search in SAT procedures. New techniques are needed to determine the diameter of a system. In particular, it would be interesting to study efficient decision procedures for QBF. Combining bounded model checking with other state space reduction techniques presents another interesting problem.

	SMV$_2$		SATO -g100 diameter		SATO -g20		PROVE diameter		PROVE	
$\|r\|$	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB
3	1	49	0	1	0	0	0	1	0	1
4	1	49	0	1	0	1	0	1	0	1
5	13	83	0	2	60	2	0	1	1	2
6	509	447	1	4	364	4	0	1	2	3
7		>1GB	3	6	1252	6	0	2	2	4
8			5	8	2160	9	0	2	7	5
9			25	14	>21h		0	3	16	9
10			42	19			1	4	55	11

Table 2. Barrel shifter ($|r|$ = number of registers, sec = seconds, MB = Mega Bytes).

	SMV$_1$		SMV$_2$		SATO $k=5$		PROVE $k=5$		SATO $k=10$		PROVE $k=10$	
cells	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB
4	846	11	159	217	0	3	1	3	3	6	54	5
5	2166	15	530	703	0	4	2	3	9	8	95	5
6	4857	18	1762	703	0	4	3	3	7	9	149	6
7	9985	24	6563	833	0	5	4	4	15	10	224	8
8	19595	31		>1GB	1	6	6	5	16	12	323	8
9	>10h				1	6	9	5	24	13	444	9
10					1	7	10	5	36	15	614	10
11					1	8	13	6	38	16	820	11
12					1	9	16	6	40	18	1044	11
13					1	9	19	8	107	19	1317	12
14					1	10	22	8	70	21	1634	14
15					1	11	27	8	168	22	1992	15

Table 3. Liveness for one user in the DME (sec = seconds, MB = Mega Bytes).

	SMV$_1$		SMV$_2$		SATO		PROVE	
cells	sec	MB	sec	MB	sec	MB	sec	MB
4	799	11	14	44	0	1	0	2
5	1661	14	24	57	0	1	0	2
6	3155	21	40	76	0	1	0	2
7	5622	38	74	137	0	1	0	2
8	9449	73	118	217	0	1	0	2
9	segmentation		172	220	0	1	1	2
10	fault		244	702	0	1	0	3
11			413	702	0	1	0	3
12			719	702	0	2	1	3
13			843	702	0	2	1	3
14			1060	702	0	2	1	3
15			1429	702	0	2	1	3

Table 4. Counterexample for liveness in a buggy DME (sec = seconds, MB = Mega Bytes).

References

[1] Arne Borälv. The industrial success of verification tools based on Stålmarck's Method. In Orna Grumberg, editor, *International Conference on Computer-Aided Verification (CAV'97)*, number 1254 in LNCS. Springer-Verlag, 1997.

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

[3] J. R. Burch, E. M. Clarke, and K. L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

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

[5] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In David L. Dill, editor, *Computer Aided Verification, 6th International Conference (CAV'94)*, volume 818 of *LNCS*, pages 415–427. Springer-Verlag, June 1994.

[6] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[7] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[8] E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time strikes back. *Science of Computer Programming*, 8:275–306, 1986.

[9] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. of the 13th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.

[10] D. S. Johnson and M. A. Trick, editors. *The second DIMACS implementation challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1993. (see `http://dimacs.rutgers.edu/Challenges/`).

[11] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. AAAI'96*, Portland, OR, 1996.

[12] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Poceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985.

[13] A. J. Martin. The design of a self-timed circuit for distributed mutual exclusion. In H. Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration*, 1985.

[14] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[15] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of Assoc. Comput. Mach.*, 32(3):733–749, 1985.

[16] G. Stålmarck and M. Säflund. Modeling and verifying systems and software in propositional logic. In B. K. Daniels, editor, *Safety of Computer Control Systems (SAFECOMP'90)*, pages 31–36. Pergamon Press, 1990.

[17] P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. Technical Report M92/112, Departement of Electrical Engineering and Computer Science, University of California at Berkley, October 1992.

[18] H. Zhang. SATO: An efficient propositional prover. In *International Conference on Automated Deduction (CADE'97)*, number 1249 in LNAI, pages 272–275. Springer-Verlag, 1997.