Bounded Model Checking — Symbolic Model Checking without BDDs

Sérgio Campos, Edmund Clarke

Problems with BDDs

- BDDs are a canonical representation. Often become too large.
- Variable ordering must be uniform along paths.
- Selecting right variable ordering very important for obtaining small BDDs.
  - Often time consuming or needs manual intervention.
  - Sometimes, no space efficient variable ordering exists.

Advantages of SAT Procedures

- SAT procedures also operate on boolean expressions but do not use canonical forms.
- Do not suffer from the potential space explosion of BDDs.
- Different split orderings possible on different branches.
- Can handle SAT problems with thousands of variables.
- Very efficient implementations available:
  - PROVER based on Stålmarck's Method.
  - SATO based on Davis-Putnam Procedure.
  - ...

Bounded Model Checking

- Bounded model checking uses a SAT procedure instead of BDDs.
- We construct boolean formula that is satisfiable iff there is a counterexample of length k.
- We look for longer and longer counterexamples by incrementing the bound k.
- After some number of iterations, we may conclude no counterexample exists and specification holds.
- For example, to verify safety properties, number of iterations is bounded by diameter of finite state machine.
- Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Amsterdam, March 1999.

Invariant and Equivalence Checking

- Invariant and equivalence checking can both be treated as special cases of bounded model checking.
- Invariant checking corresponds to the case in which the bound k is 1.
- Equivalence checking corresponds to the case in which the bound k is 0.
- Tradeoffs mentioned earlier are also reflected in SAT-based invariant and equivalence checking.

Main Advantages of Our Approach

- Bounded model checking finds counterexamples fast. This is due to depth first nature of SAT search procedures.
- It finds counterexamples of minimal length. This feature helps user understand counterexample more easily.
- It uses much less space than BDD based approaches.
- Does not need manually selected variable order or costly reordering. Default splitting heuristics usually sufficient.
- Bounded model checking of LTL formulas does not require a tableau or automaton construction.

Implementation

- Have implemented a tool BMC for our approach.
- It accepts a subset of the SMV language.
- Given k, BMC outputs a formula that is satisfiable iff counterexample exists of length k.
- If counterexample exists, SATO or PROVER (or ...) generates a truth assignment for the formula.

Performance

- Will give examples where BMC significantly outperforms BDD based model checking.
- In some cases BMC detects errors instantly, while SMV fails to construct BDD for initial state.

Definitions and Notation

- System described as a Kripke structure $M = (S, I, T, L)$, where
  - S is a finite set of states,
  - I is the set of initial states,
  - $T \subseteq S \times S$ is the transition relation, and
  - $L\colon S \to \mathcal{P}(\mathcal{A})$ is the state labeling.
- We assume every state has a successor state.

Definitions and Notation (Cont.)

- In symbolic model checking, a state is represented by a vector of state variables $s = (s(1), \ldots, s(n))$.
- We define propositional formulas $f_I(s)$, $f_T(s,t)$ and $f_p(s)$ as follows:
  - $f_I(s)$ iff $s \in I$,
  - $f_T(s,t)$ iff $(s,t) \in T$, and
  - $f_p(s)$ iff $p \in L(s)$.
- We write $T(s,t)$ instead of $f_T(s,t)$, etc.

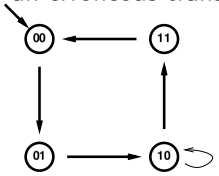
Definitions and Notation (Cont.)

- Will sometimes write $s \to t$ when $(s,t) \in T$.
- If $\pi = (s_0, s_1, \ldots)$, then $\pi(i) = s_i$ and $\pi^i = (s_i, s_{i+1}, \ldots)$.
- π is a path if $\pi(i) \to \pi(i+1)$ for all i.
- $\mathbf{E}f$ is true in M ($M \models \mathbf{E}f$) iff there is a path π in M with $\pi \models f$ and $\pi(0) \in I$.
- Model checking is the problem of determining the truth of an LTL formula in a Kripke structure. Equivalently,

  Does a witness exist for the LTL formula?

Example To Illustrate New Technique

Two-bit counter with an erroneous transition:



- Each state s is represented by state variables: $s[1]$, $s[0]$.
- In initial state, value of the counter is 0. Thus,

$$I(s) = \neg s[1] \wedge \neg s[0].$$

- Let

$$inc(s,s') = (s'[0] \leftrightarrow \neg s[0]) \wedge (s'[1] \leftrightarrow (s[0] \wedge s[1])).$$

- Define

$$T(s,s') = inc(s,s') \vee (s[1] \wedge \neg s[0] \wedge s'[1] \wedge \neg s'[0])$$

- Have deliberately added erroneous transition!!

Example (Cont.)

- Suppose we want to know if counter will eventually reach state (11).
- Can specify the property by $\mathbf{A}\,\mathbf{F}\,q$, where $q(s) = s[1] \wedge s[0]$.

  On all execution paths, there is a state where $q(s)$ holds.
- Equivalently, we can check if there is a path on which counter never reaches state (11).
- This is expressed by $\mathbf{E}\,\mathbf{G}\,p$, where $p(s) = \neg s[1] \vee \neg s[0]$.

  There exists a path such that $p(s)$ holds globally along it.

Example (Cont.)

- In bounded model checking, we consider paths of length k.
- We start with $k = 0$ and increment k until a witness is found.
- Assume k equals 2. Call the states s_0, s_1, s_2.
- We formulate constraints on s_0, s_1, and s_2 in propositional logic.
- Constraints guarantee that (s_0, s_1, s_2) is a witness for $\mathbf{E}\,\mathbf{G}\,p$ and, hence, a counterexample for $\mathbf{A}\,\mathbf{F}\,q$.

Example (Cont.)

- First, we constrain (s_0, s_1, s_2) to be a valid path starting from the initial state.
- Obtain a propositional formula

$$[\![M]\!] = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2).$$

Example (Cont.)

- Second, we constrain the shape of the path.
- The sequence of states s_0, s_1, s_2 can be a loop.
- If so, there is a transition from s_2 to the initial state s_0, s_1 or itself.
- We write ${}_lL = T(s_2, s_l)$ to denote the transition from s_2 to a state s_l where $l \in [0, 2]$.
- We define L as $\bigvee_{l=0}^{2} {}_lL$. Thus $\neg L$ denotes the case where no loop exists.

Example (Cont.)

- The temporal property $\mathbf{G}\, p$ must hold on (s_0, s_1, s_2).
- If no loop exists, $\mathbf{G}\, p$ does not hold and $[\![\mathbf{G}\, p]\!]$ is *false*.
- To be a witness for $\mathbf{G}\, p$, the path must contain a loop (condition L, given previously).
- Finally, p must hold at every state on the path

$$[\![\mathbf{G}\, p]\!] = p(s_0) \wedge p(s_1) \wedge p(s_2).$$

- We combine all the constraints to obtain the propositional formula

$$[\![M]\!] \wedge ((\neg L \wedge \mathit{false}) \vee \bigvee_{l=0}^{2} (\ {}_lL \wedge [\![\mathbf{G}\, p]\!])).$$

Example (Cont.)

- In this example, the formula is satisfiable.
- Truth assignment corresponds to counterexample path (00), (01), (10) followed by self-loop at (10).
- If self-loop at (10) is removed, then formula is unsatisfiable.

General Translation

- Given M, f and k, we construct a propositional formula $[\![M, f]\!]_k$.
- The variables $s_0, \ldots, s_k$ in $[\![M, f]\!]_k$ are states on path π.
- Each s_i is a vector of state variables.
- $[\![M, f]\!]_k$ is satisfiable iff f is valid along π.

Complexity

- Size of $[\![M, f]\!]_k$ is polynomial in size of f if common subformulas are shared.
- It is linear in k and in the size of formulas for T, I and $p \in \mathcal{A}$.
- Thus, existential bounded model checking can be reduced in polynomial time to SAT.

General Strategy

To construct $[\![M, f]\!]_k$:

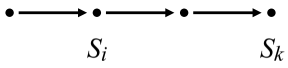
- We define a propositional formula $[\![M]\!]_k$ that constrains $s_0, \ldots, s_k$ to be on a valid path π in M.
- We give the translation of LTL formula f to a propositional formula that constrains π to satisfy f.

Unfolding the Transition Relation

For a Kripke structure M, $k \in NAT$

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

First Case–Path is Not a Loop



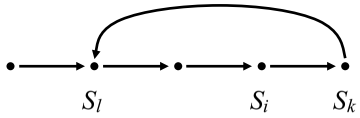
- First, we describe translation if path is not a loop.
- The translation "$[\![\cdot]\!]_k^i$" maps an LTL formula into a propositional formula.
  - k is the length of the prefix of the path that we consider.
  - i is the current position in this prefix.
- When we recursively process subformulas, i changes but k stays the same.

Translation of an LTL Formula without a Loop

f an LTL formula and $k, i \in NAT$, with $i \leq k$

$$\begin{aligned}
[\![p]\!]_k^i &:= p(s_i) \\
[\![\neg p]\!]_k^i &:= \neg p(s_i) \\
[\![f \wedge g]\!]_k^i &:= [\![f]\!]_k^i \wedge [\![g]\!]_k^i \\
[\![f \vee g]\!]_k^i &:= [\![f]\!]_k^i \vee [\![g]\!]_k^i \\
[\![\mathbf{G}\, f]\!]_k^i &:= \mathit{false} \\
[\![\mathbf{F}\, f]\!]_k^i &:= \bigvee_{j=i}^{k} [\![f]\!]_k^j \\
[\![\mathbf{X}\, f]\!]_k^i &:= \text{if } i < k \text{ then } [\![f]\!]_k^{i+1} \text{ else } \mathit{false}
\end{aligned}$$

Second Case–Path is a Loop



- Now we consider case where path is a k-loop.
- The translation "${}_l[\![\cdot]\!]^i_k$" depends on the current position i and on the length of the prefix k.
- It also depends on position l where loop starts.

Successor in a Loop

- Let $k, l, i \in NAT$, with $l, i \leq k$.
- Define the successor of i in a (k, l)-loop by

$$succ(i) := \begin{cases} i+1 & \text{for } i < k \\ l & \text{for } i = k \end{cases}$$

Translation of Formula for a Loop

f an LTL formula and $k, l, i \in NAT$, with $l, i \leq k$

$$\begin{aligned}
{}_l[\![p]\!]^i_k &:= p(s_i) \\
{}_l[\![\neg p]\!]^i_k &:= \neg p(s_i) \\
{}_l[\![f \wedge g]\!]^i_k &:= {}_l[\![f]\!]^i_k \wedge {}_l[\![g]\!]^i_k \\
{}_l[\![f \vee g]\!]^i_k &:= {}_l[\![f]\!]^i_k \vee {}_l[\![g]\!]^i_k \\
{}_l[\![\mathbf{G}\, f]\!]^i_k &:= \textstyle\bigwedge_{j=\min(i,l)}^{k} {}_l[\![f]\!]^j_k \\
{}_l[\![\mathbf{F}\, f]\!]^i_k &:= \textstyle\bigvee_{j=\min(i,l)}^{k} {}_l[\![f]\!]^j_k \\
{}_l[\![\mathbf{X}\, f]\!]^i_k &:= {}_l[\![f]\!]^{succ(i)}_k
\end{aligned}$$

Loop Condition

- Translation depends on shape of path (whether a loop or not).
- We define a loop condition to distinguish these cases. For $k, l \in NAT$, let

$$\begin{aligned}
{}_l L_k &:= T(s_k, s_l) \\
L_k &:= \bigvee_{l=0}^{k} {}_l L_k
\end{aligned}$$

General Translation

Let f be an LTL formula, M a Kripke structure, and $k \in NAT$

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

- Left side–no back loop. Use first Translation.
- Right side–path is a loop.
  - All possible start positions l of a loop are tried.
  - Translation for a (k, l)-loop is conjuncted with corresponding ${}_lL_k$ loop condition.

Correctness of Procedure

Theorem

$M \models \mathbf{E}f$ *iff* $[\![M,f]\!]_k$ *is satisfiable for some* $k \in NAT$.

Corollary

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

Sequential Multiplier Example

	SMV$_1$		SMV$_2$		SATO		PROVER	
bit	sec	MB	sec	MB	sec	MB	sec	MB
0	919	13	25	79	0	0	0	1
1	1978	13	25	79	0	0	0	1
2	2916	13	26	80	0	0	0	1
3	4744	13	27	82	0	0	1	2
4	6580	15	33	92	2	0	1	2
5	10803	25	67	102	12	0	1	2
6	43983	73	258	172	55	0	2	2
7	>17h		1741	492	209	0	7	3
8				>1GB	473	0	29	3
9					856	1	58	3
10					1837	1	91	3
11					2367	1	125	3
12					3830	1	156	4
13					5128	1	186	4
14					4752	1	226	4
15					4449	1	183	5
sum	71923		2202		23970		1066	

Model Checking: 16x16 bit sequential shift and add multiplier with overflow flag and 16 output bits.

DME Example

	SMV$_1$		SMV$_2$		SATO $k=5$		PROVER $k=5$		SATO $k=10$		PROVER $k=10$	
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

Model Checking: Liveness for one user in the DME.

"Buggy" DME Example

	SMV_1		SMV_2		SATO		PROVER	
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

Model Checking: Counterexample for liveness in a buggy DME implementation.