



Computer Aided Verification

計算機輔助驗證

Bounded Model Checking

有限模型檢驗

Pao-Ann Hsiung

Department of Computer Science and Information Engineering
National Chung Cheng University, Taiwan

熊博安

國立中正大學 資訊工程研究所



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ Experiments
- ◆ Related Work and Conclusions
- ◆ References



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ Experiments
- ◆ Related Work and Conclusions
- ◆ References



Bounded Model Checking

◆ History of Model Checking

- Explicit Model Checking
 - A few million states
 - Bottleneck: explicit enumeration of all states
- Symbolic Model Checking using OBDD
 - 10^{20} states and beyond
 - Bottleneck: exponential sizes of OBDD
- Bounded Model Checking (BMC)
 - No BDD, uses SAT techniques



Bounded Model Checking

- ◆ First proposed by Biere et al. in 1999 [1, 2]
- ◆ Does not solve complexity problem of MC
 - Still relies on an exponential procedure (SAT)
- ◆ Can solve many cases that cannot be solved by BDD-based techniques
 - Converse also true!
- ◆ Application of BMC
 - Falsification
 - Complementary to Unbounded MC (UMC)



Bounded Model Checking

- ◆ Two unique characteristics
 - User has to provide a **bound k** on the number of steps (cycles in HW) to be explored
 - Uses **SAT techniques**, instead of BDDs



Bounded Model Checking

- ◆ Basic Idea in BMC
 - Search for a **counterexample** in executions of length **bounded** by some integer k
- ◆ $k = 0, 1, 2, \dots$ until:
 - A **bug** is found, or
 - Problem becomes **intractable**, or
 - **Completeness Threshold** reached.
- ◆ BMC problem can be efficiently reduced to propositional **SAT**isfiability problem



Bounded Model Checking

- ◆ Modern SAT solvers can handle propositional satisfiability problems with **hundreds of thousands of variables** or more
- ◆ Example of SAT Solvers
 - GRASP
 - CHAFF
 - PROVER
 - SIMO
 - MATHSAT



Bounded Model Checking

◆ Experiment Results

- If k is small (60 ~ 80 cycles), depending on the model and SAT solver, BMC **outperforms** BDD-based techniques
- **Little correlation** between hard SAT problems vs. hard BDD problems
- SAT solvers can be **tuned** for BMC
- Intel verified Pentium-4™ using BMC
 - Increased **capacity** and **productivity**!



Bounded Model Checking

◆ Advantages

- Counterexamples found fast and of minimal length
- Significantly less space requirements
- No manual or dynamic reordering (as in BDD)
- Can be extended to unbounded MC
- Wide industry acceptance as soon as it was proposed
 - Intel, IBM, Compaq, ...



Bounded Model Checking

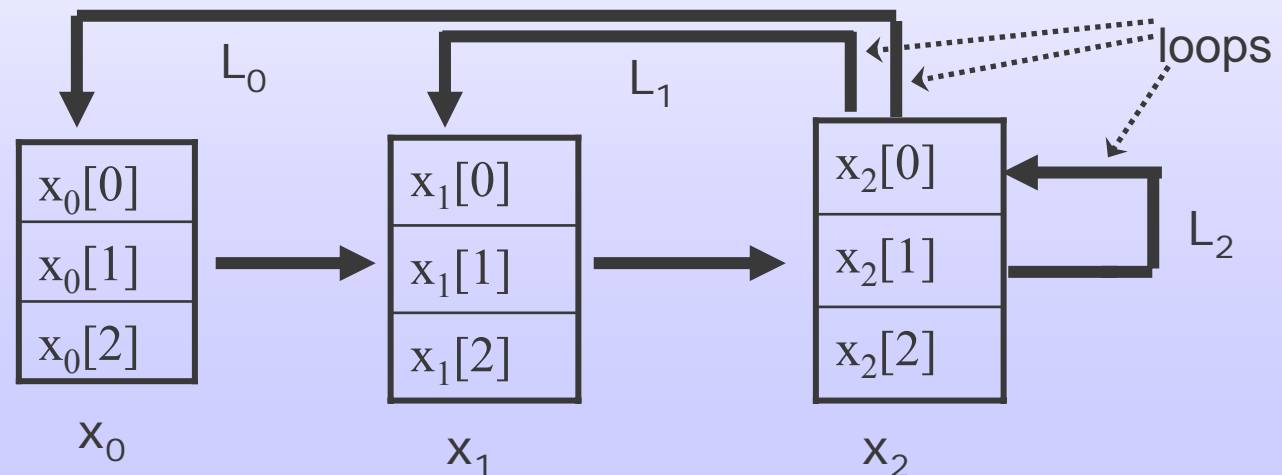
◆ Disadvantages

- Need to determine the **bound k**
- Need to be extended to UMC if a “**proof**” is required, instead of only “falsification”
- SAT solvers need to be **tuned** for BMC



BMC Example

- ◆ 3-bit shift register ($x[0]$, $x[1]$, $x[2]$)
- ◆ $T(x, x')$: $(x'[0]=x[1]) \wedge (x'[1]=x[2]) \wedge (x'[2]=1)$
- ◆ “Eventually register will be empty”: $AF(x = 0)$
- ◆ $AF(x = 0) \Leftrightarrow \neg EG(x \neq 0)$
- ◆ Restrict search to path having $k+1$ states ($k=2$)





BMC Example

- ◆ $f_m = I(x_0) \wedge T(x_0, x_1) \wedge T(x_1, x_2)$
- ◆ “Any path with three states that is a witness for $\mathbf{G}(x \neq 0)$ must contain a loop” \rightarrow add $T(x_2, x_i)$
 - Let $L_i = T(x_2, x_i)$
- ◆ Constraint imposed by the formula (S_i defined as $x_i \neq 0$) : $(x_i[0] = 1) \vee (x_i[1] = 1) \vee (x_i[2] = 1)$
- ◆ **Final Propositional Formula**
 - $f_m \wedge \bigvee_{i=0}^2 L_i \wedge \bigvee_{i=0}^2 S_i \Leftrightarrow$ Counterexample of length 2



Semantics

- ◆ **ACTL^{*}** : \subseteq CTL^{*} that are in Negative Normal Form (NNF) & contain only '**A**'s
- ◆ **ECTL^{*}**
- ◆ **LTL** : No path quantifiers are allowed
 - Consider only **X** , **F** , **G**, **U** operators
- ◆ Let's concentrate on LTL model checking
 - BMC for LTL can be extended to handle ACTL^{*} and ECTL^{*}



Semantics

- ◆ **Definition 1** : A **Kripke structure** is a tuple $M = (S, I, T, L)$ 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 $L: S \rightarrow P(A)$ with atomic propositions A
- ◆ Boolean encoding of state (vector of state variables)
- ◆ Each state has a successor state (total)
- ◆ Path $\pi = (s_0, s_1, \dots)$ $\pi(i) = s_i$ and $\pi^i = (s_i, s_{i+1}, \dots)$



Semantics

- ◆ **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 :

$\pi \models p$	iff	$p \in \ell(\pi(0))$	$\pi \models \neg p$	iff	$p \notin \ell(\pi(0))$
$\pi \models f \wedge g$	iff	$\pi \models f$ and $\pi \models g$	$\pi \models f \vee g$	iff	$\pi \models f$ or $\pi \models g$
$\pi \models \mathbf{G}f$	iff	$\forall i. \pi^i \models f$	$\pi \models \mathbf{F}f$	iff	$\exists i. \pi^i \models f$
$\pi \models \mathbf{X}f$	iff	$\pi^1 \models f$			
$\pi \models f \mathbf{U} g$	iff	$\exists i [\pi^i \models g \text{ and } \forall j, j < i. \pi^j \models f]$			
$\pi \models f \mathbf{R} g$	iff	$\forall i [\pi^i \models g \text{ or } \exists j, j < i. \pi^j \models f]$			



Semantics - Validity

- ◆ **Definition 3** : An LTL formula is **universally valid** in a Kripke structure M (in symbols $M \models Af$) 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 $M \models Ef$) iff there exists a path π in M with $\pi \models f$ and $\pi(0) \in I$
- ◆ Let's consider existential model checking problem (Search for a counterexample for EMCP)



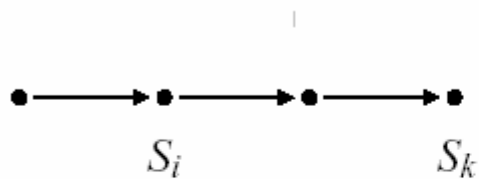
Semantics - Basic Idea of BMC

- ◆ Consider only a **finite prefix** of a path (bounded by k) and look for possible counterexample
- ◆ Finite prefix may represent an infinite path if there is a **back loop** from the last state of the prefix to any of the previous states.
- ◆ If no back loop, can't say anything about infinite behavior
- ◆ Example : Gp – Even if p holds from s_0 to s_k , can't conclude anything if there is no back loop from s_k to s_0

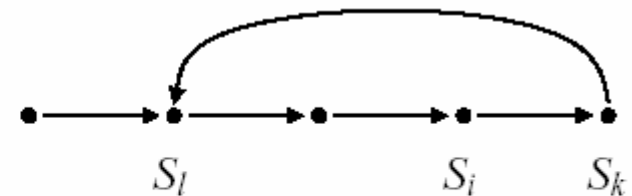


Semantics

- ◆ **Definition 4** : For $l \leq k$ we call a path π a **(k, l)-loop** if $\pi(k) \rightarrow \pi(l)$ and $\pi = u.v^\omega$ with $u = (\pi(0), \dots, \pi(l-1))$ and $v = (\pi(l), \dots, \pi(k))$. 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



(a) no loop




(b) (k, l)-loop



Semantics

- ◆ **Definition 5 (Bounded Semantics for a Loop)** : Let $k \in \mathbb{N}$ and π be a k -loop. Then an LTL formula is valid along the path π with bound k (in symbols $\pi \models_k f$) iff $\pi \models f$.
- ◆ **Definition 6 (Bounded Semantics without a Loop)** : Let $k \in \mathbb{N}$ and π be a path that is not a k -loop. Then an LTL formula is valid along the path π with bound k (in symbols $\pi \models_k f$) iff $\pi \models_k^0 f$ where:

Semantics



$\pi \models_k^i p$	<i>iff</i>	$p \in \ell(\pi(i))$	$\pi \models_k^i \neg p$	<i>iff</i>	$p \notin \ell(\pi(i))$
$\pi \models_k^i f \wedge g$	<i>iff</i>	$\pi \models_k^i f$ and $\pi \models_k^i g$	$\pi \models_k^i f \vee g$	<i>iff</i>	$\pi \models_k^i f$ or $\pi \models_k^i g$
$\pi \models_k^i \mathbf{G}f$	<i>is always false</i>				
$\pi \models_k^i \mathbf{F}f$	<i>iff</i>	$\exists j, i \leq j \leq k. \pi \models_k^j f$			
$\pi \models_k^i \mathbf{X}f$	<i>iff</i>	$i < k$ and $\pi \models_k^{i+1} f$			
$\pi \models_k^i f \mathbf{U} g$	<i>iff</i>	$\exists j, i \leq j \leq k [\pi \models_k^j g \text{ and } \forall n, i \leq n < j. \pi \models_k^n f]$			
$\pi \models_k^i f \mathbf{R} g$	<i>iff</i>	$\exists j, i \leq j \leq k [\pi \models_k^j f \text{ and } \forall n, i \leq n \leq j. \pi \models_k^n g]$			



Semantics

- ◆ **Lemma 7** : Let f be an LTL formula and π be a path and $\pi \models_k f \rightarrow \pi \models f$
- ◆ **Lemma 8** : Let f be an LTL formula and M a Kripke structure. If $M \models Ef$ then there exists $k \in \mathbb{N}$ with $M \models_k Ef$
- ◆ **Theorem 9** : Let f be an LTL formula and M a Kripke structure. Then $M \models Ef$ iff there exists $k \geq 0$ such that $M \models_k Ef$



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ Experiments
- ◆ Related Work and Conclusions
- ◆ References



Translation

- ◆ Given a Kripke structure M , LTL formula f , bound k :
 - We need to construct a **Propositional Formula** $[[M, f]]_k$ which represents the constraints on s_0, \dots, s_k (variables denoting a finite sequence of states on a path π) such that $[[M, f]]_k$ is satisfiable iff f is valid along π
 - Size $\text{poly}(f)$, $\text{quadratic}(k)$, $\text{linear}(\text{size}(\text{prop}(T, I, p \in A)))$
- ◆ **Definition 10 (Unfolding the Transition Relation)**
For a Kripke structure M , $k \in \mathbb{N}$,

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



Translation

- ◆ Depending on whether a path is a k-loop or not, two different translations for temporal formula f .
- ◆ Translation if path not a k-loop :

$$[[\cdot]]_k^i$$

- ◆ Translation if path is a k-loop :

$$_l[[\cdot]]_k^i$$

- ◆ Example : $h = p \textbf{U} q$ on a non-k-loop-path



Translation

- ◆ **Definition 11 (Translation of an LTL formula without a Loop):** For an LTL formula f and $k, i \in \mathbb{N}$

$$\begin{aligned}
 \llbracket p \rrbracket_k^i &:= p(s_i) & \llbracket \neg p \rrbracket_k^i &:= \neg p(s_i) \\
 \llbracket f \wedge g \rrbracket_k^i &:= \llbracket f \rrbracket_k^i \wedge \llbracket g \rrbracket_k^i & \llbracket f \vee g \rrbracket_k^i &:= \llbracket f \rrbracket_k^i \vee \llbracket g \rrbracket_k^i \\
 \llbracket Gf \rrbracket_k^i &:= \text{false} & \llbracket Ff \rrbracket_k^i &:= \bigvee_{j=i}^k \llbracket f \rrbracket_k^j \\
 \llbracket Xf \rrbracket_k^i &:= \text{if } i < k \text{ then } \llbracket f \rrbracket_k^{i+1} \text{ else false} \\
 \llbracket f U g \rrbracket_k^i &:= \bigvee_{j=i}^k \left(\llbracket g \rrbracket_k^j \wedge \bigwedge_{n=i}^{j-1} \llbracket f \rrbracket_k^n \right) \\
 \llbracket f R g \rrbracket_k^i &:= \bigvee_{j=i}^k \left(\llbracket f \rrbracket_k^j \wedge \bigwedge_{n=i}^j \llbracket g \rrbracket_k^n \right)
 \end{aligned}$$

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



Translation

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

$$\begin{aligned}
 {}_l\llbracket p \rrbracket_k^i &:= p(s_i) & {}_l\llbracket \neg p \rrbracket_k^i &:= \neg p(s_i) \\
 {}_l\llbracket f \wedge g \rrbracket_k^i &:= {}_l\llbracket f \rrbracket_k^i \wedge {}_l\llbracket g \rrbracket_k^i & {}_l\llbracket f \vee g \rrbracket_k^i &:= {}_l\llbracket f \rrbracket_k^i \vee {}_l\llbracket g \rrbracket_k^i \\
 {}_l\llbracket \mathbf{G}f \rrbracket_k^i &:= \bigwedge_{j=\min(i,l)}^k {}_l\llbracket f \rrbracket_k^j & {}_l\llbracket \mathbf{F}f \rrbracket_k^i &:= \bigvee_{j=\min(i,l)}^k {}_l\llbracket f \rrbracket_k^j \\
 {}_l\llbracket \mathbf{X}f \rrbracket_k^i &:= {}_l\llbracket f \rrbracket_k^{\text{succ}(i)} \\
 {}_l\llbracket f \mathbf{U} g \rrbracket_k^i &:= \bigvee_{j=i}^k \left({}_l\llbracket g \rrbracket_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l\llbracket f \rrbracket_k^n \right) \vee \\
 &\quad \bigvee_{j=l}^{i-1} \left({}_l\llbracket g \rrbracket_k^j \wedge \bigwedge_{n=i}^k {}_l\llbracket f \rrbracket_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l\llbracket f \rrbracket_k^n \right) \\
 {}_l\llbracket f \mathbf{R} g \rrbracket_k^i &:= \bigwedge_{j=\min(i,l)}^k {}_l\llbracket g \rrbracket_k^j \vee \\
 &\quad \bigvee_{j=i}^k \left({}_l\llbracket f \rrbracket_k^j \wedge \bigwedge_{n=i}^j {}_l\llbracket g \rrbracket_k^n \right) \vee \\
 &\quad \bigvee_{j=l}^{i-1} \left({}_l\llbracket f \rrbracket_k^j \wedge \bigwedge_{n=i}^k {}_l\llbracket g \rrbracket_k^n \wedge \bigwedge_{n=l}^j {}_l\llbracket g \rrbracket_k^n \right)
 \end{aligned}$$



Translation

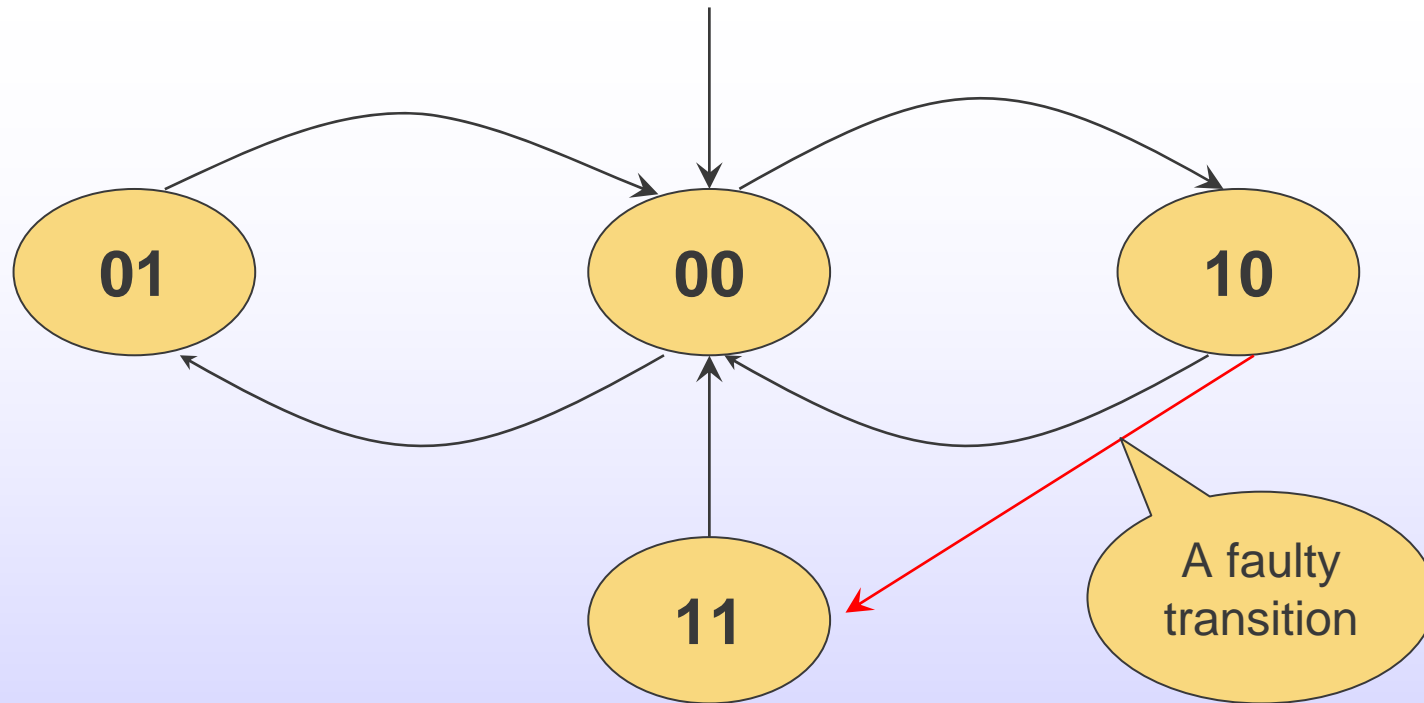
- ◆ **Definition 14 (Loop Condition)** : For $k, l \in \mathbb{N}$, let ${}_l L_k = T(s_k, s_l)$, $L_k = \bigvee_{l=0}^k {}_l L_k$
- ◆ **Definition 15 (General Translation)** : Let f be an LTL formula, M a Kripke structure and $k \in \mathbb{N}$

$$\llbracket M, f \rrbracket_k := \llbracket M \rrbracket_k \wedge \left(\left(\neg L_k \wedge \llbracket f \rrbracket_k^0 \right) \vee \bigvee_{l=0}^k \left({}_l L_k \wedge {}_l \llbracket f \rrbracket_k^0 \right) \right)$$

- ◆ **Theorem 16** : $\llbracket M, f \rrbracket_k$ is satisfiable iff $M \models_k \mathbf{E}f$
- ◆ **Corollary 17** : $M \models \mathbf{A} \neg f$ iff $\llbracket M, f \rrbracket_k$ is unsatisfiable for all $k \in \mathbb{N}$



Translation Example



Kripke structure for 2 process mutual exclusion



Translation Example

◆ Initial state

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

◆ Transition relation

$$\begin{aligned} - T(s,s') := & (\neg s[1] \wedge (s[0] \leftrightarrow \neg s'[0])) \vee \\ & (\neg s[0] \wedge (s[1] \leftrightarrow \neg s'[1])) \vee \\ & (s[0] \wedge s[1] \wedge \neg s'[1] \wedge \neg s'[0]) \end{aligned}$$

◆ Faulty transition relation

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

A faulty
transition



Translation Example

- ◆ Property to model check
 - $\mathbf{G} \neg p$, where $p = s[1] \wedge s[0]$
- ◆ Use BMC to find counterexample
 - Witness of $\mathbf{F} p$
 - Exists \rightarrow M does not satisfy $\mathbf{G} \neg p$
 - None \rightarrow M satisfies $\mathbf{G} \neg p$ up to the given bound



Translation Example

- ◆ Let bound $k = 2$
- ◆ Unrolling transition relation
 - $\llbracket M \rrbracket_2 := I(s_0) \wedge T_f(s_0, s_1) \wedge T_f(s_1, s_2)$
- ◆ Loop condition
 - $L_2 := \bigvee_{i=0,1,2} T_f(s_2, s_i)$
- ◆ Translation for paths without loops
$$\begin{aligned} \llbracket Fp \rrbracket_2^0 &:= p(s_0) \vee \llbracket Fp \rrbracket_2^1 & \llbracket Fp \rrbracket_2^1 &:= p(s_1) \vee \llbracket Fp \rrbracket_2^2 \\ \llbracket Fp \rrbracket_2^2 &:= p(s_2) \vee \llbracket Fp \rrbracket_2^3 & \llbracket Fp \rrbracket_2^3 &:= 0 \\ \llbracket Fp \rrbracket_2^0 &:= p(s_0) \vee p(s_1) \vee p(s_2) \end{aligned}$$



Translation Example

- ◆ Translation with loops can be done similarly
- ◆ Putting everything together

$$[[M, Fp]]_2 := [[M]]_2 \wedge \left(\left(\neg L_2 \wedge [[Fp]]_2^0 \right) \vee \bigvee_{i=0}^2 \left(L_2 \wedge_i [[Fp]]_2^0 \right) \right)$$

- ◆ For falsifying a safety property, loop condition can be omitted

$$\begin{aligned} [[M, Fp]]_2 &:= [[M]]_2 \wedge [[Fp]]_2^0 \\ &= I(s_0) \wedge T_f(s_0, s_1) \wedge T_f(s_1, s_2) \wedge (p(s_0) \vee p(s_1) \vee p(s_2)) \end{aligned}$$

- ◆ Assignment 00, 10, 11 satisfies $[[M, Fp]]_2$
 - Violates mutual exclusion property



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ Experiments
- ◆ Related Work and Conclusions
- ◆ References



Determining the Bound

- ◆ To check whether $M \models \mathbf{E} f$, the procedure checks $M \models_k \mathbf{E} f$ for $k = 0, 1, 2, \dots$
- ◆ 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 $M \models \mathbf{E} f$, we have to increment the value of k indefinitely, and the procedure does not terminate.



Why completeness?

- ◆ BMC may be used to clear a **module level proof obligation** which may be an **assumption** for another module
- ◆ A **missed counterexample** in a single module may **break the entire proof!**
- ◆ In such **compositional reasoning** environments, **completeness** becomes important!



Determining the Bound - ECTL

- ◆ $ECTL \subseteq ECTL^*$ with each temporal operator preceded by one 'E'
- ◆ **Theorem 18** : Given an ECTL formula f and a Kripke structure M , let $|M|$ be the number of states in M , then $M \models E f$ iff there exists $k \leq |M|$ with $M \models_k E f$



Completeness Threshold

- ◆ For every finite state system M , a property p , and a given translation scheme, there exists a number CT , such that **the absence of errors up to cycle CT proves that $M \models p$.**
- ◆ CT is the *Completeness Threshold* of M with respect to p and the translation scheme.
- ◆ For Gp formulas, CT is simply the **reachability diameter**



Determining the Bound - ECTL

- ◆ **Definition 19 (Reachability Diameter).**

Given a Kripke structure M , the reachability diameter of M is the **minimal number** $d \in \mathbb{N}$ with the following property. For every sequence of states $s_0..s_{d+1}$ with $(s_i, s_{i+1}) \in T$ for $i \leq d$, there exists a sequence of states $t_0..t_l$ where $l \leq d$ such that $t_0 = s_0$, $t_l = s_{d+1}$ and $(t_j, t_{j+1}) \in T$ for $j \leq l$.

- ◆ 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.**



Determining the Bound - ECTL

- ◆ **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:

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



Determining the Bound - ECTL

- ◆ **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 $s_0 \dots s_{d+1}$ 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 , $M \models E f$ iff there exists $k \leq d$ with $M \models_k E f$



Determining the Bound - ECTL

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

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



Determining the Bound - LTL

- ◆ LTL model checking is known to be PSPACE complete
- ◆ LTL model checking can be reduced to propositional satisfiability and thus it is in 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 $M \models_k \mathbf{E} f$.



Determining the Bound - LTL

- ◆ **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 $M \models E f$ iff there exists $k \leq u+v$ with $M \models_k E f$.



Determining the Bound - Liveness

◆ Translation of Liveness Properties

$$\llbracket M, \mathbf{AF}p \rrbracket_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \rightarrow \bigvee_{i=0}^k p(s_i)$$

◆ Theorem

$M \models \mathbf{AF}p$ iff $\exists k \llbracket M, \mathbf{AF}p \rrbracket_k$ is valid.



Determining the Bound - Liveness

- ◆ If the liveness property AFp holds, the BMC procedure terminates
 - k = length of longest sequence from initial state without hitting a state where p holds
- ◆ If AFp does not hold, then $EG\neg p$ holds, and we have a BMC procedure for $EG\neg p$ that terminates
- ◆ Does BMC is complete for liveness properties, too!



Determining the Bound - Induction

- ◆ Induction techniques for making BMC complete for safety properties
- ◆ To prove $M \models AGp$ by induction, we need to find manually a **strengthening inductive invariant**
 - An expression that
 - is **inductive** (correctness in previous step implies correctness in current step)
 - **implies the property**



Determining the Bound - Induction

- ◆ Proofs based on inductive invariants
 - Base case,
 - Induction step, and
 - Strengthening step.



Determining the Bound - Induction

- ◆ Base Case
- ◆ Given a bound n (induction depth), prove that ϕ holds in the first n steps, by checking:

$$\exists s_0, \dots, s_n. I(s_0) \wedge \bigwedge_{i=0}^{n-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^n \neg \phi(s_i)$$



Determining the Bound - Induction

- ◆ Induction Step:
- ◆ Prove the following is unsatisfiable

$$\exists s_0, \dots, s_{n+1}. \bigwedge_{i=0}^n (\phi(s_i) \wedge T(s_i, s_{i+1})) \wedge \neg \phi(s_{n+1}).$$

- ◆ Strengthening Step:
- ◆ Prove inductive invariant implies property

$$\forall s_i. \phi(s_i) \longrightarrow p(s_i)$$



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ **Propositional SAT Solvers**
- ◆ Experiments
- ◆ Related Work and Conclusions
- ◆ References



Propositional SAT Solvers

- ◆ Given a propositional formula f , a SAT solver
 - finds an **assignment** to the variables of f that satisfy it, if such an assignment exists, or
 - return '**unsatisfiable**' otherwise.



Conjunctive Normal Form (CNF)

- ◆ SAT solvers accept formulas in CNF
 - A conjunction of clauses
 - Each clause is a disjunction of literals and negated literals
- ◆ To satisfy a CNF formula, the assignment has to satisfy
 - At least ONE literal in EACH clause.



Conjunctive Normal Form (CNF)

- ◆ Every propositional formula can be transformed into CNF
 - Naïve Translation: $|CNF| = \text{exponential}(|f|)$
 - To avoid exponential size:
 - Add $O(|f|)$ auxiliary Boolean variables, where $|f|$ is the number of sub expressions in f .



Davis-Putnam Procedure

- ◆ Most modern SAT-checkers are variations of the well known Davis-Putnam procedure [5] and its improvement called DPLL [6].
- ◆ DP or DPLL Procedure
 - A back tracking search algorithm that, at each node in the search tree,
 - decides an assignment (i.e. a variable = Boolean value, which determines the next sub-tree to be traversed), and
 - computes its immediate implications by iteratively applying the 'unit clause' rule.



Davis-Putnam Procedure

- ◆ Example iteration of ‘unit clause rule’
 - If decision is $x_1=1$, then the clause $(\neg x_1 \vee x_2)$ immediately implies $x_2=1$.
 - This, in turn, can imply other assignments.
- ◆ Unit clause rule is also called “Boolean Constraint Propagation” (BCP)
- ◆ Common result of BCP
 - A clause is unsatisfiable → backtrack and change of the previous decisions



Davis-Putnam Procedure

◆ Example of BCP result

- $f: (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2),$
- decision $x_1=1 \rightarrow$
 - $x_2=1$ (applying BCP on first clause)
 - $\neg x_1 \vee \neg x_2 = 0$ (applying BCP on second clause)
- decision $x_1=1$ must be changed, and
- implications of new decision must be re-computed.



Davis-Putnam Procedure

- ◆ Backtracking
 - Pruning parts of the search tree
 - At a point of backtracking, if there are n unassigned variables
 - A sub tree of size 2^n is pruned!
- ◆ **Pruning** is the main reason why SAT is efficient!!!



Davis-Putnam Procedure

```
// Input arg: Current decision level  $d$ 
// Return value:
//   SAT(): {SAT, UNSAT}
//   Decide(): {DECISION, ALL-DECIDED}
//   Deduce(): {OK, CONFLICT}
//   Diagnose(): {SWAP, BACK-TRACK} also calculates  $\beta$ 
```

```
SAT ( $d$ )
{
 $l_1$ :   if (Decide ( $d$ ) == ALL-DECIDED) return SAT;
 $l_2$ :   while (TRUE) {
 $l_3$ :       if (Deduce( $d$ ) != CONFLICT) {
 $l_4$ :           if (SAT ( $d+1$ ) == SAT) return SAT;
 $l_5$ :           else if ( $\beta < d$  ||  $d == 0$ )
 $l_6$ :               { Erase ( $d$ ); return UNSAT; }
        }
 $l_7$ :       if (Diagnose ( $d$ ) == BACK-TRACK) return UNSAT;
    }
}
```



Davis-Putnam Procedure

- ◆ At each decision level d in the search
 - A variable assignment $V_d = \{T, F\}$ is selected with **Decide()**
 - All variables decided (**ALL-DECIDED**)
 - Return **SAT**
 - Otherwise, implied assignments are identified with **Deduce()** (BCP)
 - No conflict → recurse with higher decision level $d+1$
 - **CONFLICT** → analyze conflict with **Diagnose()**
 - Swap assignment
 - **BACK-TRACK** to decision level β (a global variable)
 - **Erase()**-ing current and all implied assignments ($d - \beta$) times



Modern SAT Checkers

- ◆ Original Davis-Putnam Procedure
 - $\beta = d - 1$ (backtracked one step at a time)
- ◆ Modern SAT checkers
 - *Non-chronological* Backtracking search strategies ($\beta = d - j, j \geq 1$)
 - Skipping a large number of irrelevant assignments
 - *Learning*
 - Adds constraints in the form of new clauses (called conflict clauses)
 - To prevent repetition of bad assignments
 - Backtracks immediate if bad assignment is repeated



Learning Example

- ◆ (a) Clause data base
- ◆ Current truth assignment $\{x_5 = 0\}$
- ◆ Current decision assignment $\{x_1 = 1\}$
- ◆ (b) Implication graph

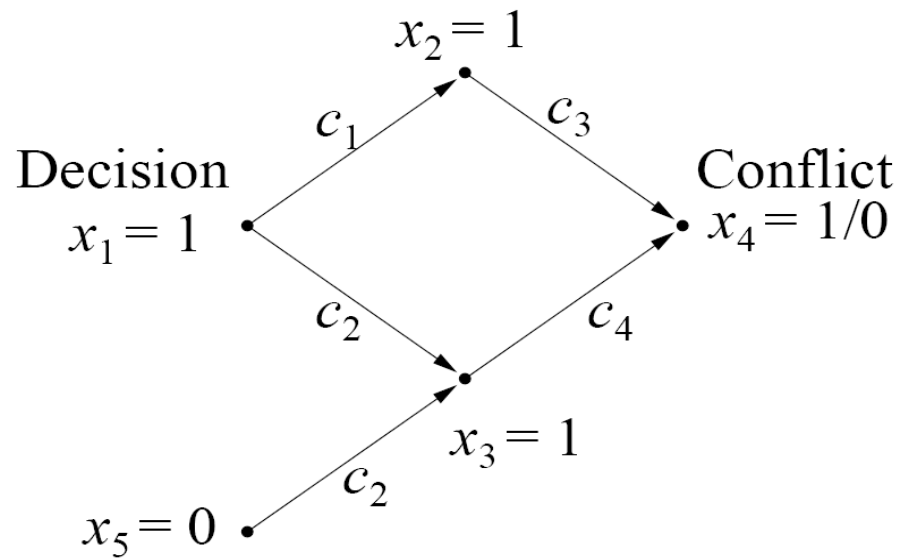
$$c_1 = (\neg x_1 \vee x_2)$$

$$c_2 = (\neg x_1 \vee x_3 \vee x_5)$$

$$c_3 = (\neg x_2 \vee x_4)$$

$$c_4 = (\neg x_3 \vee \neg x_4)$$

(a)



(b)



Learning Example

- ◆ Conflict \rightarrow either c_3 or c_4 cannot be satisfied
- ◆ Diagnose() determines the assignments directly responsible for conflict
 - $\{x_1 = 1, x_5 = 0\}$
- ◆ $(x_1 = 1) \wedge (x_5 = 0)$ gives rise to conflict
- ◆ Must ensure: $\neg((x_1 = 1) \wedge (x_5 = 0))$
 - That is, $\neg(x_1 \wedge \neg x_5) = \neg x_1 \vee x_5$
- ◆ Add new conflict clause $\pi: (\neg x_1 \vee x_5)$



New Decision Heuristics

- ◆ Decide(): strategy for picking the next variable and its value
- ◆ Order
 - **Static**: predetermined by some criterion
 - **Dynamic**: according to current state of search
 - Pick an assignment leading to largest number of satisfied clauses (**DLIS Strategy**: good decision, very large overhead)
 - Count number of times a variable occurs in a formula, newly added conflict clauses are given more weight (conflict-driven) (**Variable State Independent Decaying Sum (VSIDS) strategy**: Faster than DLIS by an order of magnitude)



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ **Experiments**
- ◆ Related Work and Conclusions
- ◆ References



Industrial Examples

- ◆ Comparing BMC with BDD-based MC
 - IBM
 - Intel
 - Compaq
 - Biere et al.
- ◆ Conclusion
 - SAT based BMC is typically faster in finding bugs compared to BDDs.
 - The deeper the bug is (i.e. the longer the shortest path leading to it is), the less advantage BMD has
 - Typical hardware design: at most 80 cycles



16x16 shift and add multiplier

- ◆ A known hard problem for BDDs
- ◆ Property
 - The output of the sequential multiplier is the same as the output of a combinational multiplier applied to the same input words
 - Verified for each of the 16 output bits separately
 - To verify bit i , sufficient to set $k = i+1$
- ◆ BDD model checker: SMV



16x16 shift and add multiplier

- ◆ Time: seconds
- ◆ Memory: MB

bit	k	SMV_2	MB	PROVER	MB
0	1	25	79	< 1	1
1	2	25	79	< 1	1
2	3	26	80	< 1	1
3	4	27	82	1	2
4	5	33	92	1	2
5	6	67	102	1	2
6	7	258	172	2	2
7	8	1741	492	7	3
8	9		>1GB	29	3
9	10			58	3
10	11			91	3
11	12			125	3
12	13			156	4
13	14			186	4
14	15			226	4
15	16			183	5

[2] DAC'1999



13 hardware designs with known bugs

- ◆ IBM's BDD model checker
 - RULEBASE₁: default configuration, with dynamic reordering
 - RULEBASE₂: without dynamic reordering, initial order from RULEBASE₁
- ◆ SAT solvers
 - GRASP: without tuning
 - GRASP (tuned): tuned for BMC
 - CHAFF: without tuning (2001)



13 hardware designs with known bugs

Model	k	RULEBASE ₁	RULEBASE ₂	GRASP	GRASP (tuned)	CHAFF
Design 1	18	7	6	282	3	2.2
Design 2	5	70	8	1.1	0.8	< 1
Design 3	14	597	375	76	3	< 1
Design 4	24	690	261	510	12	3.7
Design 5	12	803	184	24	2	< 1
Design 6	22	*	356	*	18	12.2
Design 7	9	*	2671	10	2	< 1
Design 8	35	*	*	6317	20	85
Design 9	38	*	*	9035	25	131.6
Design 10	31	*	*	*	312	380.5
Design 11	32	152	60	*	*	34.7
Design 12	31	1419	1126	*	*	194.3
Design 13	14	*	3626	*	*	9.8

Time in seconds, [7] CAV'2000



17 circuit designs from Intel

- ◆ BDD model checker
 - FORECAST
- ◆ Bounded model checker
 - THUNDER (SAT solver: SIMO)



17 circuit designs from Intel

Model	k	FORECAST (BDD)	THUNDER (SAT)
Circuit 1	5	114	2.4
Circuit 2	7	2	0.8
Circuit 3	7	106	2
Circuit 4	11	6189	1.9
Circuit 5	11	4196	10
Circuit 6	10	2354	5.5
Circuit 7	20	2795	236
Circuit 8	28	*	45.6
Circuit 9	28	*	39.9
Circuit 10	8	2487	5
Circuit 11	8	2940	5
Circuit 12	10	5524	378
Circuit 13	37	*	195.1
Circuit 14	41	*	*
Circuit 15	12	*	1070
Circuit 16	40	*	*
Circuit 17	60	*	*

Time in seconds
[8] CAV'2001



Memory system of Alpha microprocessor

- ◆ Compaq
- ◆ SAT solver
 - PROVER
- ◆ BDD model checker
 - SMV



Memory system of Alpha microprocessor

k	SMV	PROVER
25	62280	85
26	32940	19
34	11290	586
38	18600	39
53	54360	1995
56	44640	2337
76	27130	619
144	44550	10820

Time in seconds
[9] CAV'2001



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ Experiments
- ◆ **Related Work and Conclusions**
- ◆ References



Related Work

- ◆ Verification techniques based on satisfiability checking
 - Early 1990's by G. Stalmarck (Prover Technologies) based on PROVER SAT solver
 - Inductive reasoning
 - Integration with several domains achieved impressive results



Related Work

- ◆ Strichman [7] tuned SAT solvers for BMC
 - Problem-dependent variable ordering
 - Splitting heuristics
 - Pruning heuristics by exploiting regular structure of BMC formulas
 - Reusing learned information



Related Work

- ◆ BMC for Timed Systems [10]
 - MATHSAT: SAT solver extended to deal with linear constraints over real variables
 - Encoding: extends encoding for untimed systems
 - Constraints: over real variables to represent time aspects



Related Work

- ◆ SAT-based unbounded CTL model checking [McMillan CAV'02]
 - Quantifier elimination procedure
 - Top level algorithm same as BDD-based CTL model checking
 - Sets of states represented as CNF formulas, rather than with BDDs
 - Can compete with BDD-based methods and outperforms in some cases



Related Work

- ◆ SAT-based techniques used in abstraction/refinement
 - BDD-based model checker proves the abstract model
 - SAT solvers
 - Check the counterexamples to see if they are real or spurious
 - Derive refinement to abstraction



Related Work

- ◆ Structural analysis of hardware designs to derive an over approximation of the reachability diameter, thus achieving completeness.
 - Identifying frequently occurring components like memory registers, queue registers, etc.
 - Identifying SCC
 - Reachability diameter as small as 20



Conclusions

- ◆ Bounded model checking is now widely accepted by industry as a complementary tool to BDD-based model checking
- ◆ Both tools run in parallel, the first tool that finds a solution, terminates the other process



Contents

- ◆ Introduction to BMC
- ◆ Reducing BMC to SAT
- ◆ Techniques for Completeness
- ◆ Propositional SAT Solvers
- ◆ Experiments
- ◆ Related Work and Conclusions
- ◆ **References**



References

1. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," In *Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, Springer-Verlag, 1999.
2. A. Biere, A. Cimatti, E. Clarke, M. Fujita, Y. Zhu, "Symbolic Model Checking using SAT procedures instead of BDDs," In *Proc. of Design Automation Conference (DAC)*, 1999.
3. A. Biere, A. Cimatti, E. Clarke, O. Strichman, Y. Zhu, "Bounded Model Checking," *Advances in Computers*, Vol. 58, Academic Press, 2003.
4. M. Prasad, A. Biere, A. Gupta, "A Survey of Recent Advances in SAT-Based Formal Verification," *International Journal on Software Tools and Technology Transfer*, Vol. 7, No. 2, Springer, 2005.
5. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of the ACM*, Vol. 7, pp. 201-215, 1960.



References

6. M. Davis, G. Longemann, and D. Loveland, "A Machine Program for Theorem-Proving," *Communications of the ACM*, Vol. 5, pp. 394-397, 1962.
7. O. Strichman, "Tuning SAT checkers for bounded model checking," *Proc. of the 12th Intl Conference on CAV*, LNCS, Springer-Verlag, 2000.
8. F. Copt, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi, "Benefits of bounded model checking at an industrial setting," *Proc. of the 13th Intl Conference on CAV*, LNCS, pp. 436-453, 2001.
9. P. Bjesse, T. Leonard, and A. Mokkedem, "Finding bugs in an alpha microprocessor using satisfiability solvers," *Proc. of the 13 Intl Conference on CAV*, LNCS, Springer-Verlag, 2001.
10. G. Audemard, A. Cimatti, A. Kornilowicz, R. Sebastiani, "Bounded model checking for timed systems," *22nd Intl Conf. on Formal Techniques for Networked and Distributed Systems (FORTE)*, LNCS, November 2002.