SAT and Model Checking

Bounded Model Checking (BMC)  
Biere, Cimatti, Clarke, Zhu, 1999

- A.I. Planning problems: *can we reach a desired state in k steps?*
- Verification of safety properties: *can we find a bad state in k steps?*
- Verification: *can we find a counterexample in k steps?*
What is SAT?

Given a propositional formula in CNF, find if there exists an assignment to Boolean variables that makes the formula true:

$\omega_1 = (b \lor c)$
$\omega_2 = (\neg a \lor \neg d)$
$\omega_3 = (\neg b \lor d)$
$\varphi = \omega_1 \land \omega_2 \land \omega_3$

$A = \{a=0, b=1, c=0, d=1\}$

BMC idea

Given: transition system $M$, temporal logic formula $f$, and user-supplied time bound $k$

Construct propositional formula $\Omega(k)$ that is satisfiable iff $f$ is valid along a path of length $k$

Path of length $k$: $I(s_0) \land \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$

Say $f = \mathbf{EF} p$ and $k = 2$, then

$\Omega(2) = I(s_0) \land R(s_0, s_1) \land R(s_1, s_2) \land (p_0 \lor p_1 \lor p_2)$

What if $f = \mathbf{AG} p$?
BMC idea (cont’d)

\( \textbf{AG} \ p \) means \( p \) must hold in every state along any path of length \( k \)

We take

\[
-\Omega(k) = (I(s_0) \land \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})) \rightarrow \bigwedge_{i=0}^{k} p_i
\]

So

\[
\Omega(k) = I(s_0) \land \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p_i
\]

That means we look for counterexamples

Safety-checking as BMC

\( p \) is preserved up to \( k-th \) transition iff \( \Omega(k) \) is unsatisfiable:

\[
\Omega(k) = I(s_0) \land \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p
\]

If satisfiable, satisfying assignment gives counterexample to the safety property.
Example: a two bit counter

\begin{align*}
\text{Initial state: } I &= \neg l \land \neg r \\
\text{Transition: } R &= \begin{cases}
l' = (l \neq r) \land r &= -r \\
    r' &= -r
\end{cases}
\end{align*}

Safety property: \( \text{AG } (\neg l \lor \neg r) \)

\( \Omega(2) : (\neg l_0 \land \neg r_0) \land \left( l_1 = (l_0 \neq r_0) \land r_1 = \neg r_0 \land \\
    l_2 = (l_1 \neq r_1) \land r_2 = \neg r_1 \right) \land \\
\left( l_0 \land r_0 \right) \lor \\
\left( l_1 \land r_1 \right) \lor \\
\left( l_2 \land r_2 \right) \)

\( \Omega(2) \) is unsatisfiable. \( \Omega(3) \) is satisfiable.

Example: another counter

\begin{align*}
\text{I : } &\neg l \land \neg r \\
\text{R : } &\begin{cases}
l' = (l \neq r) \land r &= -r \\
    r' &= -r \\
    l &= -r
\end{cases}
\end{align*}

Liveness property: \( \text{AF } (l \land r) \)

Check: \( \text{EG } (\neg l \lor \neg r) \)

\( \Omega(2) = I(s_0) \land \bigwedge_{i=0}^{1} R(s_i, s_{i+1}) \land \bigvee_{i=0}^{2} (\neg l \lor \neg r) \land \text{loop} \)

where

\( \text{loop } = R(s_2, s_3) \land (s_3 = s_0 \lor s_3 = s_1 \lor s_3 = s_2) \)

\( \Omega(2) \) is satisfiable

Satisfying assignment gives counterexample to the liveness property
What BMC with SAT Can Do

- All LTL
- ACTL and ECTL
- In principle, all CTL and even mu-calculus
  - efficient universal quantifier elimination or
    fixpoint computation is an active area of
    research

How big should $k$ be?

- For every model $M$ and LTL property $\varphi$
  there exists $k$ s.t.
  \[ M \models_k \varphi \rightarrow M \models \varphi \]

- The minimal such $k$ is the Completeness Threshold (CT)
How big should $k$ be?

- *Diameter* $d$ = longest shortest path from an initial state to any other reachable state.
- *Recurrence Diameter* $rd$ = longest loop-free path.
- $rd \geq d$

\[
\begin{align*}
  d &= 2 \\
  rd &= 3
\end{align*}
\]

How big should $k$ be?

- *Theorem*: for $Gp$ properties $CT = d$
How big should $k$ be?

- **Theorem**: for $Fp$ properties $CT = rd$

- Open Problem: The value of $CT$ for general Linear Temporal Logic properties is unknown

A basic SAT solver

Given $\varphi$ in CNF: $(x,y,z),(-x,y),(-y,z),(-x,-y,-z)$
Basic Algorithm

While (true)
{
    if (!Decide()) return (SAT);
    while (!Deduce())
        if (!Resolve_Conflict()) return (UNSAT);
}

Choose the next variable and value. Return False if all variables are assigned.

Apply unit clause rule. Return False if reached a conflict.

Backtrack until no conflict. Return False if impossible.

DPLL-style SAT solvers

SATO, GRASP, CHAFF, BERKMIN

A = ∅

empty clause?  y  unsat

Obtain conflict clause and backtrack

conflict?  y

Branch: add some literal to A

is A total?  y  sat

n

n

n

n

n
The Implication Graph

\[ (\neg a \lor b) \land (\neg b \lor c \lor d) \]

Decisions

Assignment: \( a \land b \land \neg c \land d \)

Resolution

\[ a \lor b \lor \neg c \quad \neg a \lor \neg c \lor d \]

\[ b \lor \neg c \lor d \]

When a conflict occurs, the implication graph is used to guide the resolution of clauses, so that the same conflict will not occur again.
Conflict clauses

Assignment: \( a \land b \land \neg c \land d \)

Conflict Clauses (cont.)

- Conflict clauses:
  - Are generated by resolution
  - Are implied by existing clauses
  - Are in conflict with the current assignment
  - Are safely added to the clause set

Many heuristics are available for determining when to terminate the resolution process.
Generating refutations

- Refutation = a proof of the null clause
  - Record a DAG containing all resolution steps performed during conflict clause generation.
  - When null clause is generated, we can extract a proof of the null clause as a resolution DAG.

Unbounded Model Checking

- A variety of methods to exploit SAT and BMC for unbounded model checking:
  - Completeness Threshold
  - \( k \)-induction
  - Abstraction (refutation proofs useful here)
  - Exact and over-approximate image computations (refutation proofs useful here)
  - Use of Craig approximation
Conclusions: BDDs vs. SAT

- Many models that cannot be solved by BDD symbolic model checkers, can be solved with an optimized SAT Bounded Model Checker.
- The reverse is true as well.
- BMC with SAT is faster at finding shallow errors and giving short counterexamples.
- BDD-based procedures are better at proving absence of errors.

Acknowledgements

“Exploiting SAT Solvers in Unbounded Model Checking” by K. McMillan, tutorial presented at CAV’03

“Tuning SAT-checkers for Bounded Model Checking” and “Heuristics for Efficient SAT solving” by O. Strichman