SAT-based verification in brief

Mary Sheeran, Chalmers

SAT-based verification now hot

- Used here in Sweden since 1989 mostly in safety critical applications (railway control program verification)
- Bounded Model Checking a sensation in 1998
- SAT-based safety property verification in Lava since 1997
- Basic complete induction method described here invented by Stålmarck during a talk on inductive proofs of circuits by Koen Claessen
- SAT-based Induction and BMC used in Jasper Gold
Bounded Model Checking (BMC)

- Look for bugs up to a certain length
- Proposed for use with SAT
- Used successfully in large companies, most often for safety properties (Intel, IBM)
- Can be extended to give proofs and not just bug-finding in the particular case of safety properties. (Stålmarck et al discovered this independently of the BMC people.)
View circuit as transition system

\[(\text{dreq, q0, dack}) \rightarrow (\text{dreq}', \text{q0}', \text{dack}')\]

\[
\begin{align*}
\text{q0}' &= \text{dreq} \\
\text{dack}' &= \text{dreq} \& (\text{q0} + (\neg \text{q0} \& \text{dack}))
\end{align*}
\]
Representing transition relation as formula

\[ s = (\text{dreq}, q_0, \text{dack}) \]

\[ I(\text{dreq}, q_0, \text{dack}) = \neg q_0 & \neg \text{dack} \]

\[ T(\text{dreq}, q_0, \text{dack}, \text{dreq}', q_0', \text{dack}') \]

\[ = q_0' \leftrightarrow \text{dreq} & \text{dack}' \leftrightarrow \text{dreq} & (q_0 + (\neg q_0 & \text{dack})) \]
Composing transitions into paths

\[ \text{path}([s_0..s_i]) \]
\[ = T(s_0,s_1) \& T(s_1,s_2) \& \ldots \& T(s(i-1),s_i) \]

Representing the bad states

Similar to use of formula for initial states

\[ B(dreq,q0,dack) = dreq \& \lnot q0 \& dack \]
Transition system usually from circuit and observer

\[ B(x, y, \ldots, \text{ok}, x', y', \ldots, \text{ok}') = \neg \text{ok}' \]

Here, viewing ok as part of the state. Can also treat it specially as the only output. No major difference.
Satisfying a formula

\[ I(s_0) \text{ and path}([s_0..s_i]) \text{ and } B(s_i) \]
If system is bad

- Finds a shortest countermodel
- Error trace for debugging

But when can we stop?

when
\[
i
\]

contradictory?
Not quite, but

when

<table>
<thead>
<tr>
<th>loop-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
</tr>
</tbody>
</table>

contradictory

And symmetrically

when

<table>
<thead>
<tr>
<th>loop-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
</tbody>
</table>

contradictory
Extra formulas for loop-free

\[
\text{loop-free([s0,s1, \ldots, si])} = \quad \text{s0} \neq \text{s1} \; \& \\
\text{s0} \neq \text{s2} \; \& \\
\ldots \\
\text{s(i-1)} \neq \text{si} \quad \text{size??}
\]

States are vectors of bits, so

if \( s=(a,b,c,d) \) then

\[
\text{s0} \neq \text{s1} \quad \text{is} \quad \neg(a0 \leftrightarrow a1) \; + \\
\neg(b0 \leftrightarrow b1) \; + \\
\neg(c0 \leftrightarrow c1) \; + \\
\neg(d0 \leftrightarrow d1)
\]
A complete method

\[ i := 0 \]

\[ \text{if not Sat} \quad \text{I} \quad \text{or} \quad \text{not Sat} \quad \text{B} \]

\[ \text{then return True} \]

\[ \text{if Sat} \quad \text{I} \quad \text{B} \]

\[ \text{then return error trace} \]

\[ i := i + 1 ; \]

Another presentation

Borrowed from a presentation by Ken McMillan (about a paper I have written). See Ken’s home page for lots of good stuff!

See
http://www-cad.eecs.berkeley.edu/~kenmcml/cav03.ppt

That presentation is Copyright 2002 Cadence Design Systems
**k-induction**

induction:

\[
P(s_0) \\
\forall i: P(s_i) \Rightarrow P(s_{i+1}) \\
\forall i: P(s_i)
\]

Induction with depth k:

\[
P(s_{0..k-1}) \\
\forall i: P(s_{i..i+k-1}) \Rightarrow P(s_{i+k}) \\
\forall i: P(s_i)
\]

**k-induction with a SAT solver**

Two formulas to check:

- Base case:
  \[ I(s0) \land \text{path}([s_0...s_{k-1}]) \Rightarrow P(s_0)...P(s_{k-1}) \]

- Induction step:
  \[ \text{path}([s_0...s_k]) \land P(s_0)...P(s_{k-1}) \Rightarrow P(s_k) \]

- If both are valid, then P always holds.
- If not, increase k and try again.

Note: this is a forwards view of the problem
Simple path assumption

- Unfortunately, k-induction is not complete.
  - Some properties not k-inductive for any k.

- Simple path restriction:
  - There is a path to ¬P iff there is a simple path to ¬P (path with no repeated states).

Induction over simple paths

- Let simple\((s_{0..k})\) be defined as:
  \[ \forall i,j \text{ in } 0..k : (i \neq j) \Rightarrow s_i \neq s_j \]

- k-induction over simple paths:
  \[ \forall i: \text{simple}(s_{0..k}) \land P(s_{i..i+k-1}) \Rightarrow P(s_{i+k}) \]
  \[ \forall i: P(s_i) \]

Must hold for k large enough, since a simple path cannot be unboundedly long. Length of longest simple path is called recurrence diameter.
i:= 0

if not Sat

  i

  if not Sat
  
  then return True

  i := i+1

if Sat

  i

  if Sat
  
  then return error trace

  i := i+1

Avoid iteration from zero and tighten termination

i := some constant which can be greater than zero

if Sat

  i

  if Sat
  
  then return error trace

  i := i+1

if not Sat

  i

  if not Sat
  
  then return True

  i := i+1
Complete Method

i := some constant which can be greater than zero

if Sat

\[ \text{not (all (not B))} \]

then return error trace

if not Sat

\[ \text{all (not I)} \]

or not Sat

\[ \text{all (not B)} \]

\[ B \]

then return True

i := i+1

Strengthen

i := some constant which can be greater than zero

if Sat

\[ \text{not (all (not B))} \]

then return error trace

if not Sat

\[ \text{all (not I)} \]

then return True

i := i+1
Another way to strengthen

- Invent a lemma, $L(s)$ that we believe to hold in the reachable states
- Prove $P'(s) = P(s)$ and $L(s)$
- If both $P$ and $L$ hold in the reachable states, this can reduce induction depth

Choosing lemmas?

- Domain knowledge
- Analysis of the program
- Strongest possibility is the characterization of the reachable states

- Van Eijk’s method uses relations between signals as lemmas
Symbolic Trajectory Evaluation (STE)

\[ a \text{ is } v, \Box, \text{ c is not } v, \Box \] \rightarrow \[ \Box, \Box, \Box, \text{ d is true } \]

Symbolic simulation

- Don’t just have concrete values (and X) flowing in the circuit
- Have BDDs or formulas flowing
- A single run of a symbolic simulator checks an STE property requiring many concrete simulations
- STE is symbolic simulation plus proof that the consequent holds
Use of BMC and STE at Compaq

Aim: to automatically find violations of properties like
Same address cannot be in two entries at once
that is, bug finding during development

Reducing the problem

- Initial circuit: 400 inputs, 14 400 latches, 15 pipeline stages
- Reduced model has 10 inputs, 600 latches
Results

• Real bugs found, from 25 -144 cycles
• SAT-based BMC on 32 bit PC 20 -10k secs.
• Custom SMV on 64 bit Alpha took much longer (but goes to larger sizes)
• STE quick to run, but writing specs takes time and expertise
• Promising results in real development

NOTE: Done by Per Bjesse, who used to assist on this course 😊

Good survey

This was a Chalmers-oriented presentation :)

Most surveys find that induction does badly in comparison to other approaches to SAT-based MC. (But Safeflogic/Jasper have some secrets, which I don’t know! And induction is a corner-stone of SAT-based verification at both Intel and IBM)

For a more balanced view, read the survey paper by Prasad, Biere and Gupta (on course home page)
Future Trends

• Improved proof engines and analysis tools
• Design methods and coding rules that make necessary proofs easier
• Integration of methods
• Attempts to raise the level of abstraction
• Greater emphasis on design chains
• See Int. Roadmap for Semiconductors

References (bounded model checking)


References (safety property checking with SAT-solvers)


Niklas Eén and Niklas Sörensson. Temporal Induction by Incremental SAT-solving. BMC’03 (available on MINISAT page below. Take a look. This is great work and used all over the world.)

http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/