Model Checking with Boolean Satisfiability

Joao Marques-Silva
Electronics & Computer Science
University of Southampton

University of Bristol, February 2007
Motivation

- Remarkable improvements made to SAT solvers over the last decade
  - Clause learning; lazy data structures; adaptive branching heuristics; search restarts

- Very successful application of SAT in model checking
  - Bounded and unbounded model checking

- Existing (industry motivated) challenges
  - Ability to handle ever increasing systems
  - Ability to find deep counterexamples
  - Ability to prove difficult properties

- Lines of research
  - More efficient SAT solvers (?)
  - Better uses of SAT technology in SAT-based model checking
Outline

● SAT overview
  – Organization of a modern SAT solver
  – Resolution, resolution proofs and interpolants

● SAT-based model checking

● Improvements to SAT-based model checking

● Results & conclusions
What is SAT?

- Boolean satisfiability (SAT)
  - Given a propositional logic formula $\varphi$, decide whether there exist assignments to the propositional variables such that $\varphi$ takes value true (or 1)
    - Propositional variables: $x_1, x_2, \ldots, x_n$
    - Standard logic connectives: $\neg$, $\land$, $\lor$, $\leftrightarrow$, $\rightarrow$

  \[
  \left( x_1 \lor x_2 \right) \land \left( x_2 \rightarrow \neg x_3 \right) = 1? \]

  - If there are variable assignments that satisfy formula, then it is satisfiable; otherwise it is unsatisfiable

- Archetype NP-complete problem [Cook’71]
  - All known algorithms are exponential on the size of the representation, in the worst case
  - In practice:
    - Very effective algorithms developed over the last decade
      - Which can solve instances with millions of variables
Applications of SAT

- Many applications and many successful applications
  - Artificial Intelligence
    - Planning; Knowledge compilation; ...
  - Electronic Design Automation
    - Hardware model checking
      - Arguably the most successful application of SAT
    - Equivalence checking; Test-pattern generation; Fault diagnosis; ...
  - Software Engineering
    - Software model checking; Software testing; ...
  - Computational Biology
    - Haplotype inference; Pedigree consistency; ...
  - Theorem proving
  - Answer set programming
  - Description logics
  - ...

Extensions of SAT

- Success of SAT motivated work on extensions of SAT
  - Pseudo-Boolean Optimization (PBO) / 0-1 ILP
    - Conjunctions of linear inequalities over Boolean variables
  - Quantified Boolean Formulas (QBF)
    - Propositional logic with quantifiers
  - Satisfiability Modulo Theories
    - Decidable theories (ILA, RLA, …)
      - ILA: conjunction, disjunction and negation of linear inequalities over the integers
      - …
CNF formulas

- Conjunctive normal form (CNF):
  - Standard representation for SAT
  - CNF formula $\varphi$ is a conjunction of clauses
  - Clause is a disjunction of literals
  - Literal is a variable or its complement

$$\varphi = (a \lor b) \land (\neg a \lor c) \land (c \lor \neg d \lor \neg e) \land (\neg d \lor \neg a)$$

$$\varphi = (a \lor b)(\neg a \lor c)(c \lor \neg d \lor \neg e)(\neg d \lor \neg a)$$

- Can map propositional formulas into CNF in linear time
  - Addition of a linear number of auxiliary variables

[Tseitin'68; Plaisted&Greenbaum'86]
Given a partial assignment to the variables:

- A literal is **satisfied** if its value is 1; it is **unsatisfied** if its value is 0; otherwise it is **unassigned**
- A clause is **satisfied** if at least one of its literals is satisfied; it is **unsatisfied** if all of its literals are unsatisfied; otherwise it is **unresolved**
- A formula is **unsatisfied** if at least one clause is unsatisfied; it is **satisfied** if all clauses are satisfied; otherwise it is **unresolved**

\[ \varphi = (a \lor b) \land (\neg a \lor c) \land (c \lor \neg d \lor \neg e) \land (\neg d \lor \neg a) \]
Representing gates in CNF

\[ \varphi_d = [d = \neg(a \land b)] \]
\[ = \neg[d \oplus \neg(a \land b)] \]
\[ = \neg[\neg(a \land b) \neg d + a \land b \land d] \]
\[ = \neg[\neg a \land \neg d + \neg b \land \neg d + a \land b \land d] \]
\[ = (a + d)(b + d)(\neg a + \neg b + \neg d) \]

[Tseitin’68; Plaisted&Greenbaum’86]
Representing circuits in CNF

$\phi = h \left[ d = \neg(ab) \right] \left[ e = \neg(b+c) \right] \left[ f = \neg d \right] \left[ g = d + e \right] \left[ h = fg \right]$

$\phi$ is the characteristic function for circuit with output $h$

[Tseitin’68; Plaisted&Greenbaum’86]
Algorithms for SAT

- Incomplete Algorithms (Cannot prove unsatisfiability)
  - Local search (hill climbing)
  - Lagrangian multipliers
  - Genetic algorithms
  - Simulated annealing
  - Tabu search
  - ...

- Complete Algorithms (Can prove unsatisfiability)
  - Backtrack search (DPLL)
  - Resolution
  - Stalmarck’s method
  - Recursive learning
  - Binary decision diagrams (BDDs)
  - ...

- The utilization of SAT in model checking requires ability to prove unsatisfiability
  - Most SAT algorithms used in model checking are based on backtrack search
Plain backtrack search

- Given a CNF formula $\varphi$, i.e. a conjunction of clauses, implicitly enumerate all partial assignments to the variables.

Increasingly specified partial assignments

No variables assigned

All variables assigned

conflict: at least one unsatisfied clause

solution: all clauses satisfied
Unit propagation

- **Unit clause:**
  - A clause $\omega$ is unit iff all literals but one are assigned value 0 and one literal is unassigned
    - With $a = 0$ and $b = 1$, $\omega = (a \lor \neg b \lor c)$ is unit

- **Unit clause rule:**
  - If a clause $\omega$ is unit, then unassigned literal must be assigned value 1
    - With $a = 0$ and $b = 1$, $\omega = (a \lor \neg b \lor c)$ is unit
    - Literal $c$ must be assigned value 1 for $\omega$ to be satisfied
      - With $c = 1$, $\omega = (a \lor \neg b \lor c)$ becomes satisfied

- **Unit propagation:**
  - Iterative application of the unit clause rule
    - Imply variable assignments until no more unit clauses, or unsatisfied clause is identified

[Davis&Putnam’60]
The DPLL algorithm

- Backtrack search
  - Implicit enumeration of all partial assignments

- Unit propagation
  - Iterated application of unit clause rule

- Variable selection heuristic
  - Policy for selecting the variable to branch on and the value to assign the variable

- DPLL seldom used in practical applications until the mid 90s!
Modern SAT algorithms

- Follow the organization of the DPLL algorithm
  - Backtrack search with unit propagation

- Several key techniques are used:
  - **Clause learning** [Marques-Silva&Sakallah’96]
    - Infer new clauses from causes of conflicts
    - Allows implementing non-chronological backtracking
  - Exploiting structure of conflicts [Marques-Silva&Sakallah’96]
    - Unique Implication Points (UIPs)
    - Dominators in graph of implied assignments
  - Optimised data structures [Moskewicz et al.’01]
    - Lazy evaluation of clause state
  - Adaptive branching heuristics [Moskewicz et al.’01]
    - Variable branching metrics are affected by number of conflicts
    - Aging mechanisms for focusing on most recent conflicts
  - Search restarts [Gomes,Selman&Kautz’98]
    - Opportunistically restart backtrack search
Clause learning

During backtrack search, for each conflict learn clause that explains and prevents repetition of same conflict

\[ \varphi = (a \lor b)(\neg b \lor c \lor d)(\neg b \lor e)(\neg d \lor \neg e \lor f) \ldots \]

Assume (decisions) \( c = 0 \) and \( f = 0 \)

Assign \( a = 0 \) and imply assignments

A conflict is reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied

\[ (a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \]

\[ (\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1) \]

:. learn new clause: \( (a \lor c \lor f) \)
Non-chronological backtracking

During backtrack search, in the presence of conflicts, backtrack to one of the *causes* of the conflict.

\[ \varphi = (a \lor b)(\neg b \lor c \lor d)(\neg b \lor e)(\neg d \lor \neg e \lor f)(a \lor c \lor f)(\neg a \lor g)(\neg g \lor b)(\neg h \lor j)(\neg i \lor k) \ldots \]

Assume (decisions) \( c = 0, f = 0, h = 0 \) and \( i = 0 \)

Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learned clause \( (a \lor c \lor f) \)

\( (a \lor c \lor f) \) implies \( a = 1 \)

A conflict is again reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied

\( (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)

\( (\varphi = 1) \Rightarrow (c = 1) \lor (f = 1) \)

\[ \therefore \text{learn new clause: } (c \lor f) \]
Non-chronological backtracking

Learned clause \((c \lor f)\)

Need to backtrack, given \((c \lor f)\)

Backtrack to most recent decision: \(f = 0\)

\[\therefore \text{Clauses learned: } (a \lor c \lor f) \text{ and } (c \lor f)\]

In practice, learned clauses can allow backtracking over a significant percentage of the decision variables.
Evolution of SAT solvers

- Remarkable improvements over the last decade

<table>
<thead>
<tr>
<th>Instance</th>
<th>Posit' 94</th>
<th>Grasp' 96</th>
<th>Chaff'01</th>
<th>Siege'04</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssa2670-136</td>
<td>28.53</td>
<td>0.36</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>bf1355-638</td>
<td>772.45</td>
<td>0.04</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>design_1</td>
<td>&gt;7200</td>
<td>65.35</td>
<td>1.27</td>
<td>0.29</td>
</tr>
<tr>
<td>design_3</td>
<td>&gt;7200</td>
<td>9.13</td>
<td>0.52</td>
<td>0.41</td>
</tr>
<tr>
<td>f_ind</td>
<td>&gt;7200</td>
<td>4663.89</td>
<td>17.91</td>
<td>6.52</td>
</tr>
<tr>
<td>splitter_42</td>
<td>&gt;7200</td>
<td>&gt;7200</td>
<td>28.81</td>
<td>4.46</td>
</tr>
<tr>
<td>c6288</td>
<td>&gt;7200</td>
<td>&gt;7200</td>
<td>&gt;7200</td>
<td>2847.46</td>
</tr>
<tr>
<td>pipe_64_32</td>
<td>&gt;7200</td>
<td>&gt;7200</td>
<td>&gt;7200</td>
<td>&gt;7200</td>
</tr>
</tbody>
</table>
Resolution

- Refutation-complete procedure for first order logic

- In propositional logic:
  - Technique for deriving new clauses
    - Example: $\omega_1 = (\neg a \lor b \lor c)$, $\omega_2 = (a \lor b \lor d)$
    - Resolution:
      \[
      \text{res}(\omega_1, \omega_2, a) = (b \lor c \lor d)
      \]
  - Forms the basis of a complete procedure for satisfiability
  - Impractical for real-world formulas
  - Application of restricted forms has been successful
    - E.g., restricted resolution
      - \[\text{res}((\neg a \lor \alpha), (a \lor \alpha), a) = (\alpha)\]
      - $\alpha$ is a disjunction of literals
Resolution refutations

- Clause learning can be viewed as the inference of a clause by a sequence of resolution steps

\[ \varphi = (a \lor b)(\neg b \lor c \lor d)(\neg b \lor e)(\neg d \lor \neg e \lor f) \ldots \]
- \( a = 0 \) yields conflict; can learn \((a \lor c \lor f)\)
- By applying resolution:

\[ \varphi = (a \lor b)(\neg b \lor c \lor d)(\neg b \lor e)(\neg d \lor \neg e \lor f) \ldots \]
Deriving resolution refutations

- For unsatisfiable formulas:
  - Learned clauses capture a resolution refutation from a subset of the original clauses
  - SAT solvers can be instructed to recreate resolution refutation for unsatisfiable formula

\[
\phi = (a \lor b) \land \lnot a \lor c \land \lnot b \land \lnot c
\]

\[
\omega_1 \omega_2 \omega_3 \omega_4
\]

\[\begin{align*}
\kappa &\quad \omega_1 \quad \omega_2 \\
\omega_1 &\quad \omega_3 \\
b = 0 \quad \omega_4 \\
c = 0 \\
a = 0
\end{align*}\]

\[\begin{align*}
(a \lor b) &\quad (\lnot a \lor c) \\
(b \lor c) &\quad (\lnot b) \\
(c) &\quad (\lnot c)
\end{align*}\]
Interpolants

- Given two subsets of clauses A and B, assume \( A \land B \) is unsatisfiable. Then, there exists an interpolant \( A' \) for the pair \( (A, B) \) with the following properties:
  - \( A \) implies \( A' \)
  - \( A' \land B \) is unsatisfiable
  - \( A' \) refers only to the common variables of A and B
  - Example:
    - \( A = p \land q, B = \neg q \land r \)
    - \( A' = q \)

- Size of interpolants:
  - Given a resolution refutation of \( A \land B \), can compute interpolant for the pair \( (A, B) \) in linear time on the size of the resolution refutation
    - SAT solvers can be instructed to output resolution refutation!

- Computing interpolants:
  - Different algorithms can be used
    - Pudlak’97, McMillan’03
Computing interpolants

\[ A = (r \lor y)(\neg r \lor x) \]

\[ B = (\neg y \lor a)(\neg y \lor \neg a)(\neg x) \]

- Interpolant is a Boolean circuit that follows structure of resolution refutation
  - Can map circuit into CNF in linear time and space

[\text{Tseitin}'68; \text{Plaisted}\&\text{Greenbaum}'86]
Outline

● SAT overview

● SAT-based model checking
  – SAT-based bounded model checking (BMC)
  – Interpolant-based unbounded model checking (UMC)

● Improvements to SAT-based model checking

● Results & conclusions
Bounded model checking

- Verification of safety properties: $F \, f$

  $$\Phi^k = I_0(Y_0) \land \bigwedge_{i=0}^{k-1} T(Y_i, Y_{i+1}) \land \left( \bigvee_{i=r}^{k} f(Y_i) \right)$$

- Characteristic functions for representing initial states and transition relation, respectively $I_0$ and $T$
  - Resulting Boolean formula: $\Phi^k = I_0 \land U_k \land F_k$
    - Where:
      $$U_k = \bigwedge_{j=0}^{k-1} T_j \quad T_i = T(Y_i, Y_{i+1}) \quad F_k = \left( \bigvee_{i=r}^{k} f_i \right) \quad f_i = f(Y_i)$$
  - Interpretation:

```latex
\begin{align*}
  &I_0 \quad Y_0 & T_0 \quad Y_1 & T_1 \\
  &\cdots & \cdots & \cdots \\
  &T_{k-1} \quad Y_k & \quad F_k
\end{align*}
```
An example

- Property: $G \neg q$?
- Evaluate: $F q$
- Unroll model $k$ time steps:

Check satisfiability of CNF formula for $I_0 \land U_k \land F_k$
Bounded model checking

- A possible BMC algorithm:
  - Given some initial $k$
  - While $k \leq$ user-specified time-bound UB
    - Generate CNF formula $\varphi$ for $I_0 \land U_k \land F_k$
    - Invoke SAT solver on $\varphi$
    - If formula $\varphi$ is satisfiable, then a counterexample within $k$ time steps has been found
      - Return counterexample
    - Otherwise, increase $k$

- The BMC algorithm is incomplete
  - But complete if completeness threshold is known
Towards completeness

- Unbounded model checking
  - Utilization of induction
    - Standard BMC loop
      - Stop BMC loop for some $i$, if cannot have loop-free path of size $i$ that can be reached from $I_0$ or if cannot have loop-free path of size $i$ that can reach $F_k$
      - Maximum unfolding bounded by largest loop-free path
    - ...  
  - Utilization of interpolants
    - BMC and Craig interpolants allow SAT-based computation of abstractions of reachable states
      - Avoid computing exact sets of reachable states
      - One of the most promising approaches in practice
        - Maximum unfolding bounded by largest shortest path between any two states

[Sheeran et al.’00]  
[Chauhan et al.’02; Gupta et al.’03]  
[McMillan’03]
Abstraction of reachable states

- For each iteration of BMC loop, call to SAT solver returns unsat unless counterexample is found
  - Analysis of resolution refutation yields abstractions of reachable states
    \[ \Phi = I_0 \land T_0 \land T_1 \land \ldots \land T_{k-1} \land F_k = A \land B \]
    \[ A = I_0 \land T_0 \]
    \[ B = T_1 \land \ldots \land T_{k-1} \land F_k \]
  - Given \( A \) and \( B \), and a resolution refutation for \( A \land B \), compute Craig interpolant \( A' \):
    - \( A = I_0 \land T_0 \) implies \( A' \)
    - \( A' \land B \) is unsatisfiable
    - \( A' \) solely represented with state variables
      - If \( A \) holds, then \( A' \) holds
        - \( A_1 = A' \) represents abstraction of states reachable from \( I_0 \) in 1 time step!
Fixpoint of reachable states

- Can iterate computation of interpolants:

If \( A_i \rightarrow I_0 \lor A_1 \lor A_2 \lor ... \lor A_{i-1} \), then a fixpoint is reached; all reachable states identified!
If $F_k$ is satisfied from $I_0$, then we have a counterexample!

If a fixpoint of the reachable states is identified, then no reachable state can satisfy property!

If $A \land B$ is sat, may have abstracted too much; must unfold more time steps

Maximum value of $k$ is bounded by largest shortest path between any two states
Outline

- SAT overview
- SAT-based model checking
- Improvements to SAT-based model checking
- Results & conclusions
Rescheduling the BMC loop

$k = 0$
repeat
  if from $I_0$ can satisfy $F_k$ within $k$ steps
    return reachable
  $R = I_0$
  let $A = I_0 \land T_0$, and $B = T_1 \land T_2 \land \ldots \land T_{k-1} \land F_k$
  while $A \land B = \text{false}$
    $P = \text{unsat\_proof}(A \land B)$
    $A' = \text{interpolant}(P, A, B)$
    if $A' \rightarrow R$, return unreachable
    $R = A' \lor R$
    $A = A' \land T_0$
  end while
  increase $k$
end repeat

BMC loop

Number of iterations can be used to restrict when to check again the BMC condition!
Rescheduling the BMC loop

while $A \land B = false$

$P = \text{unsat}\_\text{proof}(A \land B)$

$A' = \text{interpolant}(P, A, B)$

if $A' \rightarrow R$, return unreachable

$R = A' \land R$

$A = A' \land T_0$

end while

Fixpoint checking with $i+1$ iterations (last iteration is sat):

$I_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_{i+1}$

Checked all states reachable in up to $k+i$ states, with an unfolding of size $k$; no counterexample was found

.. Need to check BMC condition only when unfolding of FSM exceeds $k+i$ time steps

In general useful if counterexample exists
Results on rescheduling

- Evaluated rescheduling on different benchmarks
  - Specifically designed and industrial examples
- Evaluated both the plain UMC algorithm and rescheduling

<table>
<thead>
<tr>
<th>Instance</th>
<th>No-reschedule</th>
<th>Reschedule BMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit counter</td>
<td>0.31</td>
<td>0.09</td>
</tr>
<tr>
<td>5-bit counter</td>
<td>3.86</td>
<td>0.84</td>
</tr>
<tr>
<td>6-bit counter</td>
<td>21.36</td>
<td>10.41</td>
</tr>
<tr>
<td>7-bit counter</td>
<td>1780.68</td>
<td>175.69</td>
</tr>
<tr>
<td>I12</td>
<td>255.77</td>
<td>272.47</td>
</tr>
<tr>
<td>I11</td>
<td>75.28</td>
<td>81.89</td>
</tr>
<tr>
<td>I31</td>
<td>83.51</td>
<td>90.08</td>
</tr>
<tr>
<td>I32</td>
<td>19.66</td>
<td>14.89</td>
</tr>
<tr>
<td>I33</td>
<td>17.44</td>
<td>13.09</td>
</tr>
<tr>
<td>I21</td>
<td>24.93</td>
<td>26.48</td>
</tr>
<tr>
<td>Total time</td>
<td>2282.8</td>
<td>685.9</td>
</tr>
</tbody>
</table>
Conclusions

● SAT technology has improved dramatically over the last decade
  – Key techniques:
    • Clause learning, optimized data structures, adaptive branching heuristics, search restarts

● SAT has been applied to model checking with success
  – Bounded and unbounded model checking

● Optimisations to the use of interpolants