Boolean Satisfiability: From Theoretical Hardness to Practical Success

Sharad Malik
Princeton University
SAT in a Nutshell

- Given a Boolean formula, find a variable assignment such that the formula evaluates to 1, or prove that no such assignment exists.

\[ F = (a + b)(a' + b' + c) \]

- For \( n \) variables, there are \( 2^n \) possible truth assignments to be checked.

- First established NP-Complete problem.

Where are we today?

- Intractability of the problem no longer daunting
  - Can regularly handle practical instances with millions of variables and constraints

- SAT has matured from theoretical interest to practical impact
  - Electronic Design Automation (EDA)
    - Widely used in many aspects of chip design
  - Increasing use in software verification
    - Commercial use at Microsoft, NEC,…
  - CAV 2009 Award for industrial impact
Expensive Bugs Drive Hardware Verification

- Intel Pentium FDIV Bug 1994
  - incorrect results for division
  - due to errors in the entries in the lookup table used by the digital divide operation algorithm
- Total cost associated with replacement: $475 million
SAT Solvers in the Verification Context

- (static) program verification
- interpolation
- resolution
- complexity
- predicate abstraction
- BMC
- equivalence checking
- symbolic simulation
- SMT
- decision procedures
- compiler correctness
- model checking
- temporal logic
- proof theory
Welcome to SAT Live!

If you are a newcomer to the SATisfiability problem, you might want to take a look at [wikipedia's page on the boolean satisfiability problem](http://en.wikipedia.org/wiki/Boolean_satisfiability_problem) first. You might also find those surveys on the current interest in SAT solvers for software and hardware verification insightful. Armin Biere's course on formal systems is a good start. Eugene Goldberg has also written a nice article on practical applications of boolean satisfiability.

The SAT association also makes available some chapters from the Handbook of Satisfiability to allow newcomers to understand key principles in satisfiability, namely History of Satisfiability. Learning architecture in SAT solvers and the principles behind Bounded Model Checking. More material is available on the association's tutorials web page.

Looking for a SAT solver to play with? The following open-source SAT solvers might be a good start: Minisat (C++), Picosat (C), SAT4J (Java). If you are looking for a stochastic local search solver, take a look at UBCSAT.

You can take a look at all the current links, see the links classified by keywords, or add your own reference (you must be subscribed to SAT Live or propose it as anonymous).

If you don't have any links to propose for now but would like email notification of new additions to the repository, you can subscribe to the SAT Live notification list or register to the site's MUSE, using Dapper.

Finally, a page with some people interested by the SATisfaction problem is also available.

**Last 10 new entries**

<table>
<thead>
<tr>
<th>Date</th>
<th>Title</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-Apr-2013</td>
<td>Google Summer of Code Opportunity</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Contributed by: Daniel Le Berre</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keywords: General Interest</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The computer science and engineering group of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the Vienna University of Technology has</td>
<td></td>
</tr>
<tr>
<td></td>
<td>been accepted as a mentoring organization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for the 3rd consecutive year in Google's</td>
<td></td>
</tr>
<tr>
<td></td>
<td>program (<a href="http://www.google-melange.org/gsoc/">http://www.google-melange.org/gsoc/</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>org/google/gsoc2013/cse_tuwien), and Skeptik</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<a href="http://github.com/Paradoxik/Skeptik">http://github.com/Paradoxik/Skeptik</a>) is one</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of its projects. Skeptik is a tool for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>checking, compressing and improving formal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>proofs generated by automated deduction tools</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and the Skeptik-SMT solver. It implements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>various recent proof compression algorithms,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>such as RecycleProof, RecycleProofWith</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction, Split, Reduce &amp; Reconstruct,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LowerUnits, as combinations and variants of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>these algorithms. Students interested in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>applying for a Google Summer of Code grant</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(US$ 5000) to work from 17th of June to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23rd of September on the development of new</td>
<td></td>
</tr>
<tr>
<td></td>
<td>proof combs in Skeptik should follow the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>instructions in this web page:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/Paradoxik/Skeptik/wiki/G">https://github.com/Paradoxik/Skeptik/wiki/G</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>soc-Instructions. The &quot;DEADLINE&quot; for student</td>
<td></td>
</tr>
<tr>
<td></td>
<td>applications is 3rd of May 2013. Before</td>
<td></td>
</tr>
<tr>
<td></td>
<td>preparing their applications, students are</td>
<td></td>
</tr>
<tr>
<td></td>
<td>encouraged to contact us (preferably by</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sending a message to our mailing list</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<a href="http://jwein2.iue.tuwien.ac.at:8088/mail/">http://jwein2.iue.tuwien.ac.at:8088/mail/</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>with &quot;(skeptic)&quot; in the subject).</td>
<td></td>
</tr>
</tbody>
</table>
SAT Competition 2013

affiliated with the SAT 2013 conference, July 8-12 in Helsinki, Finland.
Jointly organized by Iim University, University of Helsinki, University College Dublin and University of Texas at Austin.

About the SAT Competition 2013

The SAT Competition 2013 is a competitive event for solvers for the Boolean Satisfiability (SAT) problem. It is organized as a satellite event to the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013) and stands in the tradition of the SAT Competitions that have been held yearly from 2002 to 2005 and biennially starting from 2007, the SAT-Races held in 2006, 2008 and 2010, and the SAT Challenge 2012.

The emphasis of SAT Competition 2013 is on evaluation of core solvers. Additionally, and new this year, the UNSAT Tracks of the competition will require certification.

News

Apr 12 All deadlines are postponed by 3 days. Please check new deadlines
Apr 10 Solver description template with specific instructions available.
Feb 22 Calls for Participation and Benchmarks.
Feb 21 Website is up.

Website template modified from a design by Arash
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **Extensions and Applications**
  - MAX SAT, Correction Sets, Backbones and Debugging
Problem Representation

- **Conjunctive Normal Form**
  - Representation of choice for modern SAT solvers

\[(a+b+c)(a'+b'+c)(a'+b+c')(a+b'+c')\]

- Variables
- Literals
- Clauses
Circuit to CNF Conversion

- **Tseitin Transformation**

\[ d \equiv (a + b) \]
\[ (a + b + d') \]
\[ (a' + d) \]
\[ (b' + d) \]

\[ e \equiv (c \cdot d) \]
\[ (c' + d' + e) \]
\[ (d + e') \]
\[ (c + e') \]

Consistency conditions for circuit variables

- **Can ‘e’ ever become true?**

Is \((e)(a + b + d')(a'+d)(b'+d)(c'+d+e)(d+e')(c+e')\) satisfiable?
Resolution

- Resolution of a pair of distance-one clauses

\[(a + b + c' + f) \quad (g + h' + c + f)\]

Resolvent implied by the original clauses
Davis Putnam Algorithm


- Iterative existential quantification of variables

Potential memory explosion problem!
SAT Solvers: A Condensed History

- Deductive
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- Backtrack Search
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- Conflict Driven Clause Learning [CDCL]
  - GRASP: Integrate a constraint learning procedure, 1996

- Locality Based Search
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- Extensions and Applications
  - MAX SAT, Correction Sets, Backbones and Debugging
Basic DLL Search

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(b' + c' + d)\]
\[(a' + b + c')\]
\[(a' + b' + c)\]

Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)

(a' + b + c')
(a' + b' + c)
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

→

Decision
Basic DLL Search

\[(a' + b + c)\]  
\[(a + c + d)\]  
\[(a + c + d')\]  
\[(a' + b + c)\]  
\[(a' + b + c')\]  
\[(a' + b' + c)\]  
\[(a' + b' + c')\]  

Diagram:

- Node a
- Node b
- Node c
- Arrows pointing from a to b to c
- Decision point at c
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Unit Clause Rule

Implication Graph
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
&\quad \rightarrow \quad (a + c + d) \\
&\quad \quad \rightarrow \quad (a + c + d') \\
&\quad \quad \quad \rightarrow \quad (a + c' + d) \\
&\quad \quad \quad \quad \rightarrow \quad (a + c' + d') \\
&\quad \quad \quad \quad \quad \rightarrow \quad (a' + b + c) \\
&\quad \quad \quad \quad \quad \quad \rightarrow \quad (a' + b + c') \\
&\quad \quad \quad \quad \quad \quad \quad \rightarrow \quad (a' + b' + c) \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \rightarrow \quad (a' + b' + c')
\end{align*}
\]

Implication Graph

\[
\begin{align*}
d=1, \quad d=0
\end{align*}
\]
Basic DLL Search

Implication Graph

\[a' + b + c\]
\[\rightarrow\]
\[a + c + d\]
\[\rightarrow\]
\[a + c + d'\]
\[a + c' + d\]
\[a + c' + d'\]
\[b' + c' + d\]
\[a' + b + c'\]
\[a' + b' + c\]

\[d = 1, d = 0\] Conflict!
Basic DLL Search

\[(a' + b + c) \rightarrow (a + c + d) \rightarrow (a + c + d') \rightarrow (a + c' + d) \rightarrow (a + c' + d') \rightarrow (b' + c' + d) \rightarrow (a' + b + c') \rightarrow (a' + b' + c)\]
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Implication Graph

Conflicting paths:
- (a' + c' + d)
- (a + c' + d')

Forced Decision

d=1, d=0
Basic DLL Search

\[
\begin{align*}
&(a' + b + c) \\
\rightarrow&(a + c + d) \\
\rightarrow&(a + c + d') \\
\rightarrow&(a + c' + d) \\
\rightarrow&(a + c' + d') \\
\rightarrow&(b' + c' + d) \\
\rightarrow&(a' + b + c') \\
&\quad(a' + b' + c)
\end{align*}
\]
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' + d') \\
(b' + c' + d) \\
(a' + b + c') \\
(a' + b' + c)
\end{align*}
\]

\[
0 
\quad \quad 
1 \quad \leftarrow \text{Forced Decision}
\]

Diagram:
- Node a with edges to nodes b and c
- Node b with edges to nodes 0 and 1
- Node c with edges to nodes 0 and 1
- Red boxes at the bottom
Basic DLL Search

\[(a' + b + c)\]
\[\rightarrow (a + c + d)\]
\[\rightarrow (a + c + d')\]
\[\rightarrow (a' + c' + d)\]
\[\rightarrow (a' + c' + d')\]
\[\rightarrow (b' + c' + d)\]
\[\rightarrow (a' + b + c')\]
\[\rightarrow (a' + b' + c)\]

\[d=1\]
\[d=0\]

\[c=0\]

\[\text{Conflict!}\]

\[\text{Implication Graph}\]

\[a=0\]
\[\text{Conflict!}\]

\[c=0\]
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' + d') \\
(b' + c' + d) \\
(a' + b + c') \\
(a' + b' + c)
\end{align*}
\]

Diagram:

```
          a
         / \  \\
        0   b
         / 1  \\
        c  c
```

Backtrack

```
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)
```
Basic DLL Search

(a' + b + c)  
(a + c + d)  
(a + c + d')  
(b' + c' + d)  
(a' + b + c')  
(a' + b' + c)

Implication Graph

Conflict!

Forced Decision

d=1, d=0
Basic DLL Search

\[ (a' + b + c) \]
\[ (a + c + d) \]
\[ (a + c + d') \]
\[ (a + c' + d) \]
\[ (a + c' + d') \]
\[ (b' + c' + d) \]
\[ (a' + b + c') \]
\[ (a' + b' + c) \]
Basic DLL Search

\[ \begin{align*}
(a' + b + c) &
(a + c + d) \\
(a + c + d') &
(a + c' + d) \\
(a + c' + d') &
(b' + c' + d) \\
(a' + b + c') &
(a' + b' + c) \\
\end{align*} \]

\[ \begin{array}{c}
\rightarrow \\
\rightarrow \\
\rightarrow \\
\rightarrow \\
\end{array} \]

\[ \begin{align*}
0 &
1 \\
0 &
1 \\
0 &
1 \\
\end{align*} \]

\[ \Rightarrow \text{ Forced Decision} \]
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Implication Graph
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Implication Graph

(a' + b' + c)  (b' + c' + d)

a=1  c=1  d=1
b=1

→

(a' + b + c)

b

0 1

→
c

0 1

0 1

→
a

0 1

1

→
c=1,d=1
Basic DLL Search

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(b' + c' + d)\]
\[(a' + b + c')\]
\[(a' + b' + c)\]

\[\text{Implication Graph}\]

\[\text{c}=1, \text{d}=1\]

\[\text{SAT}\]
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **Extensions and Applications**
  - MAX SAT, Correction Sets, Backbones and Debugging
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
Conflicting Driven Learning and Non-chronological Backtracking

\[
\begin{align*}
&x_1 + x_4 \\
&x_1 + x_3' + x_8' \\
&x_1 + x_8 + x_{12} \\
&x_2 + x_{11} \\
&x_7' + x_3' + x_9 \\
&x_7' + x_8 + x_9' \\
&x_7 + x_8 + x_{10'} \\
&x_7 + x_{10} + x_{12'}
\end{align*}
\]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]

\[ x_4 = 1 \]
\[ x_1 = 0, x_4 = 1 \]
\[ x_3 = 1 \]
Conflict Driven Learning and Non-chronological Backtracking

\begin{align*}
& x_1 + x_4 \\
& x_1 + x_3' + x_8' \\
& x_1 + x_8 + x_12 \\
& x_2 + x_{11} \\
& x_7' + x_3' + x_9 \\
& x_7' + x_8 + x_9' \\
& x_7 + x_8 + x_{10'} \\
& x_7 + x_{10} + x_{12'} \\
\end{align*}

\begin{itemize}
  \item \( x_1 = 0, \ x_4 = 1 \)
  \item \( x_3 = 1, \ x_8 = 0 \)
  \item \( x_1 = 0 \)
  \item \( x_3 = 1 \)
  \item \( x_8 = 0 \)
\end{itemize}
Conflict Driven Learning and Non-chronological Backtracking

\[
\begin{align*}
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10'} \\
x_7 + x_{10} + x_{12'}
\end{align*}
\]
Conflict Driven Learning and Non-chronological Backtracking

\[\begin{align*}
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10}' \\
x_7 + x_{10} + x_{12}'
\end{align*}\]
Conflict Driven Learning and Non-chronological Backtracking

\[ \begin{align*}
&x_1 + x_4 \\
&x_1 + x_3' + x_8' \\
&x_1 + x_8 + x_{12} \\
&x_2 + x_{11} \\
&x_7' + x_3' + x_9 \\
&x_7' + x_8 + x_9' \\
&x_7 + x_8 + x_{10'} \\
&x_7 + x_{10} + x_{12'} \\
\end{align*} \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]

\[ x_4 = 1 \]
\[ x_1 = 0 \]
\[ x_3 = 1 \]
\[ x_7 = 1 \]
\[ x_11 = 1 \]
\[ x_2 = 0 \]
\[ x_8 = 0 \]
\[ x_{12} = 1 \]
Conflict Driven Learning and Non-chronological Backtracking

- $x_1 + x_4$
- $x_1 + x_3' + x_8'$
- $x_1 + x_8 + x_{12}$
- $x_2 + x_{11}$
- $x_7' + x_3' + x_9$
- $x_7' + x_8 + x_9'$
- $x_7 + x_8 + x_{10}'$
- $x_7 + x_{10} + x_{12}'$

```
 x1 = 0, x4 = 1
 x3 = 1, x8 = 0, x12 = 1
 x2 = 0, x11 = 1
 x7 = 1, x9 = 0, 1
```

Diagram:

- $x_4 = 1$
- $x_1 = 0$
- $x_3 = 1$
- $x_7 = 1$
- $x_9 = 1$
- $x_9 = 0$
- $x_8 = 0$
- $x_{11} = 1$
- $x_{12} = 1$
- $x_2 = 0$
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]

Add conflict clause: \( x_3' + x_7' + x_8 \)
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

Add conflict clause: \[ x_3' + x_7' + x_8 \]

\[ x_1=0, x_4=1 \]
\[ x_3=1, x_8=0, x_{12}=1 \]
\[ x_2=0, x_{11}=1 \]
\[ x_7=1, x_9=1 \]
\[ x_3=1 \land x_7=1 \land x_8=0 \rightarrow \text{conflict} \]

Add conflict clause: \[ x_3' + x_7' + x_8 \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]

Backtrack to the decision level of \( x_3=1 \)
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12'} \]
\[ x_3' + x_7' + x_8 \] ← new clause

Backtrack to the decision level of \( x_3 = 1 \)
Assign \( x_7 = 0 \)
What’s the big deal?

Conflict clause: $x_1' + x_3 + x_5'$

- Significantly prune the search space – learned clause is useful forever!
- Useful in generating future conflict clauses.
Restart

- Abandon the current search tree and reconstruct a new one
- The clauses learned prior to the restart are still there after the restart and can help pruning the search space
- Adds to robustness in the solver

Conflict clause: $x_1' + x_3 + x_5'$
Unsatisfiable Cores

- An unsatisfiable core is an inconsistent subset of the clauses of the original formula:

$$\left( \overline{s} \right) \left( \overline{r} + s \right) \left( r \right) \left( s \right)$$

- An unsatisfiable core is minimal if dropping one of its clauses makes it satisfiable.

- Can be extracted from resolution proof.

Extracting Unsatisfiable Cores

Resolution Proof of Unsatisfiability

Empty Clause

Original Clauses

Learned Clauses
Extracting Unsatisfiable Cores

Resolution Proof of Unsatisfiability

Original Clauses
Learned Clauses
Empty Clause
Extracting Unsatisfiable Cores

Resolution Proof of Unsatisfiability

- Core Clauses
- Original Clauses
- Learned Clauses

Empty Clause
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **Extensions and Applications**
  - MAX SAT, Correction Sets, Backbones and Debugging
Success with Chaff

- First major instance: Tough (Industrial Processor Verification)
  - Bounded Model Checking, 14 cycle behavior

- Statistics
  - 1 million variables
  - 10 million literals initially
    - 200 million literals including added clauses
    - 30 million literals finally
  - 4 million clauses (initially)
    - 200K clauses added
  - 1.5 million decisions
  - 3 hour run time

Avoid expensive book-keeping for unit-propagation

N-literal clause can be unit or conflicting only after N-1 of the literals have been assigned to F

\[(v_1 + v_2 + v_3): \text{implied cases: } (0 + 0 + v_3) \text{ or } (0 + v_2 + 0) \text{ or } (v_1 + 0 + 0)\]

Can completely ignore the first N-2 assignments to this clause

Pick two literals in each clause to “watch” and thus can ignore any assignments to the other literals in the clause.

Example: \((v_1 + v_2 + v_3 + v_4 + v_5)\)

\[(v_1=X + v_2=X + v_3=? \{\text{i.e. } X \text{ or } 0 \text{ or } 1\} + v_4=? + v_5=?\)\]

Maintain the invariant: If a clause can become newly implied via any sequence of assignments, then this sequence will include an assignment of one of the watched literals to F
For every clause, two literals are watched.

- When a variable is assigned true, only need to visit clauses where its watched literal is false (only one polarity)
  - Pointers from each literal to all clauses it is watched in
- In a $n$ clause formula with $v$ variables and $m$ literals
  - Total number of pointers is $2n$
  - On average, visit $n/v$ clauses per assignment
- *No updates to watched literals on backtrack*
“Assign most tightly constrained variable”: e.g. DLIS (Dynamic Largest Individual Sum)

- Simple and intuitive: At each decision simply choose the assignment that satisfies the most unsatisfied clauses.
- Expensive book-keeping operations required
  - Must touch *every* clause that contains a literal that has been set to true. Often restricted to initial (not learned) clauses.
  - Need to reverse the process for un-assignment.

- Look ahead algorithms even more compute intensive
  
  C. Li, Anbulagan, “Look-ahead versus look-back for satisfiability problems”
  

- Take a more “global” view of the problem
Chaff Contribution 2: Activity Based Decision Heuristics

- **VSIDS: Variable State Independent Decaying Sum**
  - Rank variables by literal count in the initial clause database
  - Only increment counts as new (learnt) clauses are added
  - Periodically, divide all counts by a constant

- **Quasi-static:**
  - Static because it doesn’t depend on variable state
  - Not static because it gradually changes as new clauses are added
    - Decay causes bias toward *recent* conflicts.
    - Has a beneficial interaction with 2-literal watching
Activity Based Heuristics and Locality Based Search

- By focusing on a sub-space, the covered spaces tend to coalesce
  - More opportunities for resolution since most of the variables are common.
  - Variable activity based heuristics lead to locality based search
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **Extensions and Applications**
  - MAX SAT, Correction Sets, Backbones and Debugging
MAX SAT

- Maximum number of clauses that can be satisfied
- Given an UNSAT instance

\[(\overline{r} + \overline{s} + t) (\overline{r} + s) (r) (s) (\overline{t})\]

- What is the largest subset of clauses that can be satisfied?
Partial MAX SAT

- Maximum number of clauses that can be satisfied given certain clauses can’t be dropped
  - Correspond to hard constraints
- Given an UNSAT instance

\[(\overline{r} + \overline{s} + t) (\overline{r} + s) (r) (s) (\overline{t})\]

Which clauses do we have to drop to make it satisfiable?
(the pinned clauses can’t be dropped)
Minimal Correction Set (MCS)

- Minimal Correction Set:
  - Minimal set of clauses which need to be dropped to make formula satisfiable
  - Minimal: No subset makes the formula satisfiable

- The complement of any MAX-SAT solution is an MCS
- Converse doesn’t hold: Complement of MCS is maximal set of satisfiable clauses
Minimal Correction Set (MCS)

- The following formula has a single minimum \textit{um} MCS:

\[
\{ \overline{s} \}, \: (\overline{r} + s), \: (r), \: (s) \]

- \{ (r), (s) \} is minimal but not minimum.
- The formula has three different MCSes.
Applications: Hardware Verification

- Hardware verification
- Enabling technique for Bounded Model Checking

Can be encoded as Boolean Formula

Huffman Model \[ N = f(P, I) \]
Applications: Bounded Model Checking

- Analyze fixed number of execution steps/cycles
  - Verification of temporal logic properties
    (e.g., “eventually every request is acknowledged”)

```
\begin{array}{c}
\text{C_1} \\
\text{C_2} \\
\text{C_3} \\
\text{C_4}
\end{array}
```

```
\begin{array}{c}
\text{s_1} \\
\text{s_2} \\
\vdots \\
\text{s_j}
\end{array}
```

```
\begin{array}{c}
\text{t_1} \\
\text{t_2} \\
\vdots \\
\text{t_j}
\end{array}
```

```
\begin{array}{c}
\text{i_1^1 \ldots i_m^1} \\
\text{i_1^2 \ldots i_m^2} \\
\text{i_1^3 \ldots i_m^3} \\
\text{i_1^4 \ldots i_m^4}
\end{array}
```

```
\begin{array}{c}
\text{o_1^1 \ldots o_n^1} \\
\text{o_1^2 \ldots o_n^2} \\
\text{o_1^3 \ldots o_n^3} \\
\text{o_1^4 \ldots o_n^4}
\end{array}
```
Design Debugging: Fault Localization

- Circuit formula constrained by specification is inconsistent
  - Detects error
- Manually **localizing** the fault causing a known error is tedious

![Diagram of a circuit with nodes labeled 1 and 2, showing input and output values.]
Fault Localization Using MCSes

- Use MCSes to identify error location
- Input/output values are hard constraints (we’re not interested in MCSes including them)

\[
(\overline{s}) (i^{1}_{1}) (i^{1}_{2}) (o^{1}) \quad (\overline{t}) (i^{2}_{1}) (i^{2}_{2}) (o^{2})
\]

\[
D
\]

- cycle 1
  \[
  (\overline{r} + i^{1}_{1}) \cdot (\overline{r} + s) \cdot (i^{1}_{1} + \overline{s} + r)
  \]
- cycle 2
  \[
  (\overline{t} + i^{2}_{1}) \cdot (\overline{t} + r) \cdot (i^{2}_{1} + \overline{r} + t)
  \]

\[
D
\]

- cycle 1
  \[
  (i^{1}_{2} + o^{1}) \cdot (\overline{s} + o^{1}) \cdot (o^{1} + i^{1}_{2} + s)
  \]
- cycle 2
  \[
  (i^{2}_{2} + o^{2}) \cdot (\overline{r} + o^{2}) \cdot (o^{2} + i^{2}_{2} + r)
  \]
Debugging Hardware Prototypes

- Long traces
  - Long separation between fault excitation and error observation
- Limited observability of signals in manufactured chip
  - Trace buffers: Limited recording of select signals
  - Scan chains: Read-out after chip execution stopped
Hardware Faults and MCSes

- Analysis limited to small (contiguous) sequence of cycles
- Scalability of decision procedure determines window size
- Slide window backwards in time to cover different cycles
Inferring “Fixed” Signals for Satisfiable Instances

- Backbone of a satisfiable formula:
  Set of variables that have same value in all satisfying assignments

- Consider the satisfiable formula

\[ (r \oplus t) \cdot (r + s) \cdot (r) \]

- Satisfying assignments:

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Propagating Backbones

- Across Sliding Windows
  - Backbones provide a specific abstraction

Charlie Shucheng Zhu, Georg Weissenbacher and Sharad Malik,
Post-Silicon Fault Localisation Using Maximum Satisfiability and Backbones,
Concluding Remarks

- SAT: Significant shift from theoretical interest to practical impact.
- Quantum leaps between generations of SAT solvers
- Successful application of diverse CS techniques
  - Logic (Deduction and Solving), Search, Caching, Randomization, Data structures, efficient algorithms
  - Engineering developments through experimental computer science
- Presence of drivers results in maximum progress.
  - Electronic design automation – primary driver and main beneficiary
  - Software verification - the next frontier
- Opens attack on even harder problems
  - SMT, Max-SAT, QBF...

References

References


References


[HJS08] Youssef Hamadi, Said Jabbour, and Lakhdar Sais, ManySat: solver description, Microsoft Research-TR-2008-83
