A brief introduction to practical SAT solving

What can I expect from SAT today?

Daniel Le Berre

Joint work with Anne Parrain, Pascal Rapicault, Olivier Roussel,
Laurent Simon, and others

CRIL-CNRS UMR 8188 - Université d’Artois

SAT Workshop, 24–25 February 2011, Tübingen
Context: SAT receives much attention since a decade

Why are we all here today?

- Most companies doing software or hardware verification are now using SAT solvers.
- SAT technology indirectly reaches our everyday life:
  - Intel core i7 processor designed with the help of SAT solvers [Kaivola et al, CAV 2009]
  - Windows 7 device drivers verified using SAT related technology (Z3, SMT solver) [De Moura and Bjorner, IJCAR 2010]
  - The Eclipse open platform uses SAT technology for solving dependencies between components [Le Berre and Rapicault, IWOCE 2009]
- Many SAT solvers are available from academia or the industry.
- SAT solvers can be used as a black box with a simple input/output language (DIMACS).
- The consequence of a new kind of SAT solver designed in 2001 (Chaff)
Outline

SAT in theory

SAT in practice: the international SAT competition

SAT solvers for solving “real” problems: the CDCL architecture

Nearby SAT: PBO, MaxSAT, MUS

Using boolean constraints to solve Software Dependency problems

Conclusion

Depending on time: a quick overview of Sat4j
The SATisfiability problem

Definition
Input: A set of clauses built from a propositional language with $n$ variables.
Output: Is there an assignment of the $n$ variables that satisfies all those clauses?
The SATisfiability problem

**Definition**
Input: A set of clauses built from a propositional language with \( n \) variables.
Output: Is there an assignment of the \( n \) variables that satisfies all those clauses?

**Example**

\[
C_1 = \{ \neg a \lor b, \neg b \lor c \} = (\neg a \lor b) \land (\neg b \lor c) = (a' + b)(b' + c)
\]

\[
C_2 = C_1 \cup \{ a, \neg c \} = C_1 \land a \land \neg c
\]

For \( C_1 \), the answer is yes, for \( C_2 \) the answer is no

\[
C_1 \models \neg (a \land \neg c) = \neg a \lor c
\]
Where are clauses coming from?
My own background : Automated Reasoning/Artificial Intelligence

Suppose:

- a: I like beer
- b: I should visit Germany
- c: I should drink German beer

Then $C_1$ could represent the beliefs:

- $a \implies b$: If I like beer, then I should visit Germany.
- $b \implies c$: If I visit Germany, then I should drink German beer.

What happens if I like beer and I do not drink German beer ($a \land \neg c$)? This is inconsistent with my beliefs.

From $C_1$ I can deduce $a \implies c$: If I like beer, then I should drink German beer.
Definition
Given an initial state $s_0$, a state transition relation $ST$, a goal state $g$ and a bound $k$.
Is there a way to reach $g$ from $s_0$ using $ST$ within $k$ steps?
Is there a succession of states $s_0, s_1, s_2, \ldots, s_k = g$ such that
$\forall \ 0 \leq i < k \ (s_{i-1}, s_i) \in ST$ ?

- The problems are generated for increasing $k$.
- For small $k$, the problems are usually UNSATISFIABLE
- For larger $k$, the problems can be either SAT or UNSAT.
- Complete SAT solvers are needed!
1992 - Planning As Satisfiability

\[
PAS(S, I, T, G, k) = I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} G(s_i)
\]

where:

- \( S \) the set of possible states \( s_i \);
- \( I \) the initial state;
- \( T \) transitions between states;
- \( G \) goal state;
- \( k \) bound.

If the formula is satisfiable, then there is a plan of length \( k \).
\[ SMA(S, \text{op}, p) = \exists s, s' \in S \ \text{op}(s, s') \land p(s) \land \neg p(s') \]

where:
- \( S \) the set of possible states
- \( \text{op} \) an operation
- \( p \) an invariant

If the formula is satisfiable, then there is an execution of the operation that breaks the invariant.
BMC\((S, I, T, p, k) = I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p(s_i)\)

where:

- \(S\) the set of possible states \(s_i\)
- \(I\) the initial state
- \(T\) transitions between states
- \(p\) is an invariant property
- \(k\) a bound

If the formula is satisfiable, then there is a counter-example reachable in \(k\) steps.
SAT is important in theory ...

- Canonical NP-Complete problem (Cook, 1971)
- Threshold phenomenon on randomly generated $k$-SAT instances (Mitchell, Selman, Levesque, 1992)

Source: http://www.isi.edu/~szekely/antsebook/ebook/modeling-tools-and-techniques.htm
... but theory does not always meet practice

- Cannot translate a CNF into 3-CNF and use ratio \( \frac{\text{#clauses}}{\text{#variables}} \) to know in advance if a SAT instance is difficult to solve or not.

- Adding more variables can speed up solving time (despite theoretical lower and upper bounds complexity depending on the number of variables \( n \), e.g. \( 1.473^n \)).

- Adding more clauses (redondent ones) may also speed up the solving time.

- Solving real problems with SAT solvers requires specific expertise.

- How to find out which approaches work well in practice?
... but theory does not always meet practice

- Cannot translate a CNF into 3-CNF and use ratio $\frac{\#\text{clauses}}{\#\text{variables}}$ to know in advance if a SAT instance is difficult to solve or not.
- Adding more variables can speed up solving time (despite theoretical lower and upper bounds complexity depending on the number of variables $n$, e.g. $1.473^n$).
- Adding more clauses (redundant ones) may also speed up the solving time.
- Solving real problems with SAT solvers requires specific expertise.
- How to find out which approaches work well in practice?
- ... give it a try! :)'
... but theory does not always meet practice

- Cannot translate a CNF into 3-CNF and use ratio $\frac{\#\text{clauses}}{\#\text{variables}}$ to know in advance if a SAT instance is difficult to solve or not.
- Adding more variables can speed up solving time (despite theoretical lower and upper bounds complexity depending on the number of variables $n$, e.g. $1.473^n$).
- Adding more clauses (redondent ones) may also speed up the solving time.
- Solving real problems with SAT solvers requires specific expertise.
- How to find out which approaches work well in practice?
  - ... give it a try! :)
  - ... the international SAT competition or SAT Race
Outline

SAT in theory

**SAT in practice : the international SAT competition**

SAT solvers for solving “real” problems : the CDCL architecture

Nearby SAT : PBO, MaxSAT, MUS

Using boolean constraints to solve Software Dependency problems

Conclusion

Depending on time : a quick overview of Sat4j
In the present paper, a uniform proof procedure for quantification theory is given which is feasible for use with some rather complicated formulas and which does not ordinarily lead to exponentiation. The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore’s routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using the present method in 30 minutes [Davis and Putnam, 1960].

The well-formed formula (...) which was beyond the scope of Gilmore’s program was proved in under two minutes with the present program [Davis et al., 1962]
A brief history of SAT competitive events
Since the beginning the SAT community has been keen to run competitive events

- 1992 : Paderborn
- 1993 : The second DIMACS challenge [standard input format]
- 1996 : Beijing
- 1999 : SATLIB
- 2000 : SAT-Ex
- Since 2002 : yearly competition (or Race)
DIMACS common input format

```
p cnf 3 4
-1 2 0
-2 3 0
1 0
-3 0
```

- Not really fun!
- Designed for SAT solver designers, not end users!
  - Need to know in advance the number of variables and clauses (p cnf line)
  - Use only integer to denote variables

SAT technology cannot be used directly by end users (they need a more user friendly layer/API)
To keep in mind before looking at the results of a competition

Results depends on:

▶ available solvers and benchmarks.
▶ hardware (amount of RAM, size of L2 cache, etc).
▶ operating system (linux)
▶ competition rules (timeout, source code)
▶ the amount of computing resources available
▶ ...

We do not claim to have statistically meaningful results!
Impressive results of the SAT 2009 competition

General information on the benchmark

<table>
<thead>
<tr>
<th>Name</th>
<th>APPLICATIONS/c32sat/post-cbmc-zfcp-2.8-a2-nholes.cnf</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDSSUM</td>
<td>c4aa2ddc6b0ee7687df95a32ceed5b43</td>
</tr>
<tr>
<td>Bench Category</td>
<td>APPLICATION (applications instances)</td>
</tr>
<tr>
<td>Best result obtained on this benchmark</td>
<td>SAT</td>
</tr>
<tr>
<td>Best CPU time to get the best result obtained on this benchmark</td>
<td>51.6012</td>
</tr>
<tr>
<td>Satisfiable</td>
<td></td>
</tr>
<tr>
<td>(Un)Satisfiability was proved</td>
<td></td>
</tr>
<tr>
<td>Number of variables</td>
<td>10950109</td>
</tr>
<tr>
<td>Number of clauses</td>
<td>32697160</td>
</tr>
<tr>
<td>Sum of the clauses size</td>
<td>76320846</td>
</tr>
<tr>
<td>Maximum clause length</td>
<td>65</td>
</tr>
<tr>
<td>Minimum clause length</td>
<td>1</td>
</tr>
<tr>
<td>Number of clauses of size 1</td>
<td>2415</td>
</tr>
<tr>
<td>Number of clauses of size 2</td>
<td>21783823</td>
</tr>
<tr>
<td>Number of clauses of size 3</td>
<td>10900782</td>
</tr>
<tr>
<td>Number of clauses of size 4</td>
<td>1892</td>
</tr>
<tr>
<td>Number of clauses of size 5</td>
<td>131</td>
</tr>
<tr>
<td>Number of clauses of size over 5</td>
<td>1307</td>
</tr>
</tbody>
</table>

Results of the different solvers on this benchmark

<table>
<thead>
<tr>
<th>Solver Name</th>
<th>TraceID</th>
<th>Answer</th>
<th>CPU time</th>
<th>Wall clock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAppertioT_base (complete)</td>
<td>1563400</td>
<td>SAT</td>
<td>51.6012</td>
<td>204.435</td>
</tr>
<tr>
<td>picosat 913 (complete)</td>
<td>1563397</td>
<td>SAT</td>
<td>216.724</td>
<td>120.088</td>
</tr>
<tr>
<td>adaptg2wsat2009 2009-03-23 (incomplete)</td>
<td>1563429</td>
<td>?</td>
<td>0</td>
<td>0.00836907</td>
</tr>
<tr>
<td>Hybrid2 2009-03-22 (Incomplete)</td>
<td>1563418</td>
<td>?</td>
<td>0</td>
<td>0.0509205</td>
</tr>
<tr>
<td>TNM 2009-03-22 (Incomplete)</td>
<td>1563420</td>
<td>?</td>
<td>0</td>
<td>0.0590984</td>
</tr>
<tr>
<td>iPAWS 2009-03-22 (Incomplete)</td>
<td>1563407</td>
<td>?</td>
<td>0.000999</td>
<td>0.00850804</td>
</tr>
<tr>
<td>adaptg2wsat2009+ 2009-03-23 (incomplete)</td>
<td>1563430</td>
<td>?</td>
<td>0.000999</td>
<td>0.00578992</td>
</tr>
<tr>
<td>NCVWr 2009-03-22 (Incomplete)</td>
<td>1563419</td>
<td>?</td>
<td>0.000999</td>
<td>0.00562898</td>
</tr>
<tr>
<td>MXC 2009-03-10 (complete)</td>
<td>1563395</td>
<td>?</td>
<td>2.40463</td>
<td>2.42289</td>
</tr>
</tbody>
</table>
Impressive results of the SAT 2009 competition

<table>
<thead>
<tr>
<th>Satisfiable</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Un)Satisfiability was proved</td>
<td></td>
</tr>
<tr>
<td>Number of variables</td>
<td>10950109</td>
</tr>
<tr>
<td>Number of clauses</td>
<td>32697150</td>
</tr>
<tr>
<td>Sum of the clauses size</td>
<td>76320846</td>
</tr>
<tr>
<td>Maximum clause length</td>
<td>65</td>
</tr>
<tr>
<td>Minimum clause length</td>
<td>1</td>
</tr>
<tr>
<td>Number of clauses of size 1</td>
<td>2415</td>
</tr>
<tr>
<td>Number of clauses of size 2</td>
<td>21783823</td>
</tr>
<tr>
<td>Number of clauses of size 3</td>
<td>10907882</td>
</tr>
<tr>
<td>Number of clauses of size 4</td>
<td>1592</td>
</tr>
<tr>
<td>Number of clauses of size 5</td>
<td>131</td>
</tr>
<tr>
<td>Number of clauses of size over 5</td>
<td>1307</td>
</tr>
</tbody>
</table>

Different solvers on this benchmark

<table>
<thead>
<tr>
<th>Solver Name</th>
<th>TraceID</th>
<th>Answer</th>
<th>CPU time</th>
<th>Wall clock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SApperIoT base (complete)</td>
<td>1563400</td>
<td>SAT</td>
<td>51.5012</td>
<td>204.435</td>
</tr>
<tr>
<td>picosat 913 (complete)</td>
<td>1563397</td>
<td>SAT</td>
<td>116.704</td>
<td>120.088</td>
</tr>
<tr>
<td>adaptg2wsat2009 2009-03-23 (incomplete)</td>
<td>1563429</td>
<td>?</td>
<td>0</td>
<td>0.00536097</td>
</tr>
</tbody>
</table>
Impressive results of the SAT 2009 competition

General information on the benchmark

<table>
<thead>
<tr>
<th>Name</th>
<th>CRAFTEDlegen/unsat/sgen1-unsat121-100.cnf</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD6SUM</td>
<td>2488650a2f97ba63d116c240cb3c6b69f</td>
</tr>
<tr>
<td>Bench Category</td>
<td>CRAFTED (crafted instances)</td>
</tr>
<tr>
<td>Best result obtained on this benchmark</td>
<td></td>
</tr>
<tr>
<td>Best CPU time to get the best result obtained on this benchmark</td>
<td></td>
</tr>
<tr>
<td>Satisfiable</td>
<td></td>
</tr>
<tr>
<td>(Un)Satisfiability was proved</td>
<td></td>
</tr>
<tr>
<td>Number of variables</td>
<td>121</td>
</tr>
<tr>
<td>Number of clauses</td>
<td>252</td>
</tr>
<tr>
<td>Sum of the clauses size</td>
<td>756</td>
</tr>
<tr>
<td>Maximum clause length</td>
<td>3</td>
</tr>
<tr>
<td>Minimum clause length</td>
<td>3</td>
</tr>
<tr>
<td>Number of clauses of size 1</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 2</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 3</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 4</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 5</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size over 5</td>
<td>0</td>
</tr>
</tbody>
</table>

Results of the different solvers on this benchmark

<table>
<thead>
<tr>
<th>Solver Name</th>
<th>TraceID</th>
<th>Answer</th>
<th>CPU time</th>
<th>Wall clock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT07 reference solver: SATzilla CRAFTED (complete)</td>
<td>1785609</td>
<td>4996.65</td>
<td>5011.1</td>
<td></td>
</tr>
<tr>
<td>SATzilla2009_1 2009-03-22 (complete)</td>
<td>1825787</td>
<td>(exit code)</td>
<td>4996.65</td>
<td>5000.32</td>
</tr>
<tr>
<td>VARSAT-Industrial 2009-03-22 (complete)</td>
<td>1785604</td>
<td>? (TO)</td>
<td>5000.04</td>
<td>5001.93</td>
</tr>
<tr>
<td>glucose 1.0 (complete)</td>
<td>1784160</td>
<td>? (TO)</td>
<td>5000.06</td>
<td>5002.51</td>
</tr>
<tr>
<td>iut_bmb_sat 1.0 (complete)</td>
<td>1785601</td>
<td>? (TO)</td>
<td>5000.06</td>
<td>5002.21</td>
</tr>
<tr>
<td>mxc 2009-03-10 (complete)</td>
<td>1785416</td>
<td>? (TO)</td>
<td>5000.08</td>
<td>5001.81</td>
</tr>
<tr>
<td>Sappero1 base (complete)</td>
<td>1785605</td>
<td>? (TO)</td>
<td>5000.06</td>
<td>5001.51</td>
</tr>
<tr>
<td>MiniSat 2.1 (Sat-race 08 Edition)</td>
<td>1784159</td>
<td>? (TO)</td>
<td>5000.1</td>
<td>5002.21</td>
</tr>
<tr>
<td>clasp 1.2.0-SAT09-32 (complete)</td>
<td>1785600</td>
<td>? (TO)</td>
<td>5000.1</td>
<td>5013.71</td>
</tr>
<tr>
<td>procset 2.6 (complete)</td>
<td>1784158</td>
<td>? (TO)</td>
<td>5000.1</td>
<td>5002.11</td>
</tr>
</tbody>
</table>
Impressive results of the SAT 2009 competition

<table>
<thead>
<tr>
<th>Satisfiable</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Un)Satisfiability was proved</td>
<td></td>
</tr>
<tr>
<td>Number of variables</td>
<td>121</td>
</tr>
<tr>
<td>Number of clauses</td>
<td>252</td>
</tr>
<tr>
<td>Sum of the clauses size</td>
<td>756</td>
</tr>
<tr>
<td>Maximum clause length</td>
<td>3</td>
</tr>
<tr>
<td>Minimum clause length</td>
<td>3</td>
</tr>
<tr>
<td>Number of clauses of size 1</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 2</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 3</td>
<td>252</td>
</tr>
<tr>
<td>Number of clauses of size 4</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size 5</td>
<td>0</td>
</tr>
<tr>
<td>Number of clauses of size over 5</td>
<td>0</td>
</tr>
</tbody>
</table>

Different solvers on this benchmark

<table>
<thead>
<tr>
<th>Solver Name</th>
<th>TraceID</th>
<th>Answer</th>
<th>CPU time</th>
<th>Wall clock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT07 reference solver: SATzilla CRAFTED (complete)</td>
<td>1785603</td>
<td>? (exit code)</td>
<td>4998.65</td>
<td>5001.1</td>
</tr>
<tr>
<td>SATzilla2009_C 2009-03-22 (complete)</td>
<td>1825787</td>
<td>? (exit code)</td>
<td>4998.65</td>
<td>5000.32</td>
</tr>
<tr>
<td>VARSAT-industrial 2009-03-22 (complete)</td>
<td>1785604</td>
<td>? (TO)</td>
<td>5000.04</td>
<td>5001.91</td>
</tr>
<tr>
<td>grules 1.0 (complete)</td>
<td>1784182</td>
<td>0 (TO)</td>
<td>5000.04</td>
<td>5000.51</td>
</tr>
</tbody>
</table>
Huge success in the community!

The chart shows the number of solvers submitted over the years from 2002 to 2009. The data is categorized into two groups: CDCL and Others. The years are labeled on the x-axis, and the number of solvers submitted is on the y-axis. The bars are color-coded, with CDCL in blue and Others in red.

- **2002**: CDCL (5) and Others (20)
- **2003**: CDCL (14) and Others (14)
- **2004**: CDCL (18) and Others (37)
- **2005**: CDCL (21) and Others (23)
- **2007**: CDCL (22) and Others (22)
- **2009**: CDCL (31) and Others (19)

The chart visually represents the growth in solver submissions with a notable increase in 2004 and a steady increase towards 2009.
Understanding “Cactus Plots”

Same data, “probabilistic” view
Outline

SAT in theory

SAT in practice: the international SAT competition

SAT solvers for solving “real” problems: the CDCL architecture

Nearby SAT: PBO, MaxSAT, MUS

Using boolean constraints to solve Software Dependency problems

Conclusion

Depending on time: a quick overview of Sat4j
Which solver to choose for solving “real” SAT problems

1990s  Local Search
Fast boolean enumeration, built-in randomization
Main application : Planning as Satisfiability

2000s  Conflict Driven Clause Learning
Fast Conflict Analysis, Cheap clause learning penalty,
adaptive heuristics
Main application : Bounded Model Checking

2010s  ???
My guess : moving to optimization problems, not
simply decision problems
The breakthrough: Chaff

- 2 order of magnitude speedup on unsat instances compared to existing approaches on BMC (Velev) benchmarks.
- Immediate speedup for SAT based tools: BlackBox "Supercharged with Chaff"
- Based on careful analysis of GRASP internals [Marques-Silva and Sakallah, 1996]
- 3 key features:
  - New lazy data structure: Watched literals
  - New adaptative heuristic: Variable State Independent Decaying Sum
  - New conflict analysis approach: First UIP
- Taking into account randomization
The Conflict Driven Clause Learning approach

CDCL is not heuristic binary search!
Another way to represent how it works

Go from 0000000.... to 1111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

▶ Proceed from left to right
▶ Decisions are represented by 0
▶ Propagations are represented by 1
▶ When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

0
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

01
Another way to represent how it works

Go from 0000000... to 1111111... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

011
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

0111
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

▶ Proceed from left to right
▶ Decisions are represented by 0
▶ Propagations are represented by 1
▶ When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

01110
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

011101
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

0111011
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

01110110
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

0111011011
Another way to represent how it works

Go from 0000000.... to 1111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

01110110111
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

01110110111101 conflict
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

0111011011110
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

011101111011111011111 conflict
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

011101111011111011111
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

01110111101111111111111111 conflict
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

0111011110111111111111111
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

▶ Proceed from left to right
▶ Decisions are represented by 0
▶ Propagations are represented by 1
▶ When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

11011110111011110 conflict
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

... and so on until either...
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

11110111101111101111011 sat
Another way to represent how it works

Go from 0000000.... to 11111111..... such that

- Proceed from left to right
- Decisions are represented by 0
- Propagations are represented by 1
- When a conflict occurs, backtrack, i.e. go back to a 0 and change it by a 1

11111111111111111111111 unsat
Chaff : a highly coupled set of features

- Learning does not degrade solver performance thanks to the watched literals.
- The VSIDS heuristics does not need a complete picture of the reduced formula, i.e. is compatible with the lazy data structure.
- VSIDS take advantage of the conflict analysis to spot important literals.
- VSIDS provides different orders of literals at each restart.
- VSIDS adapt itself to the instance!
Since then ...

- Minisat simplified architecture, heuristic scheme, preprocessor and conflict minimization strategy.
- Specific data structures for binary and ternary clauses (Siege?)
- Phase caching (RSAT) and Rapid restarts (many authors and solvers)
- Aggressive learned clauses deletion strategy? (Glucose)
- Dynamic restarts (many authors)
- ...

...
The reason of the success?

- Better engineering (level 2 cache awareness)?
The reason of the success?

- Better engineering (level 2 cache awareness)?
- Better tradeoff between speed and intelligence?
The reason of the success?

- Better engineering (level 2 cache awareness)?
- Better tradeoff between speed and intelligence?
- Instance-based auto adaptation?
The reason of the success?

- Better engineering (level 2 cache awareness)?
- Better tradeoff between speed and intelligence?
- Instance-based auto adaptation?
- ...

29/60
The reason of the success?

▶ Better engineering (level 2 cache awareness)?
▶ Better tradeoff between speed and intelligence?
▶ Instance-based auto adaptation?
▶ ...

All those reasons are correct.
But there is a more fundamental reason too ...
CDCL has a better proof system than DPLL!
Proof theory strikes back!

- ... thanks to many others before ...
Definition

p-simulation Proof system S p-simulates proof system T, if, for every unsatisfiable formula \( \varphi \), the shortest refutation proof of \( \varphi \) in S is at most polynomially longer than the shortest refutation proof of \( \varphi \) in T.

Theorem 1 [Pipatsrisawat, Darwiche 09]. CLR with any asserting learning scheme p-simulates general resolution.
Outline

SAT in theory

SAT in practice: the international SAT competition

SAT solvers for solving “real” problems: the CDCL architecture

Nearby SAT: PBO, MaxSAT, MUS
  MaxSat
  Pseudo-Boolean Optimization
  MUS

Using boolean constraints to solve Software Dependency problems

Conclusion
Associate to each constraint (clause) a weight (penalty) \( w_i \) taken into account if the constraint is violated: Soft constraints \( \phi \).

Special weight (\( \infty \)) for constraints that cannot be violated: hard constraints \( \alpha \).

Find a model \( I \) of \( \alpha \) that minimizes \( \text{weight}(I, \phi) \) such that:

- \( \text{weight}(I, (c_i, w_i)) = 0 \) if \( I \) satisfies \( c_i \), else \( w_i \).
- \( \text{weight}(I, \phi) = \sum_{w_c \in \phi} \text{weight}(I, w_c) \)

<table>
<thead>
<tr>
<th>Weight</th>
<th>( \infty )</th>
<th>denomination</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \infty )</td>
<td>yes</td>
<td>Sat</td>
</tr>
<tr>
<td>( k )</td>
<td>no</td>
<td>MaxSat</td>
</tr>
<tr>
<td>( k )</td>
<td>yes</td>
<td>Partial MaxSat</td>
</tr>
<tr>
<td>( N )</td>
<td>no</td>
<td>Weighted MaxSat</td>
</tr>
<tr>
<td>( N )</td>
<td>yes</td>
<td>Weighted Partial MaxSat</td>
</tr>
</tbody>
</table>
Extending SAT 2: Pseudo-Boolean problems

Linear Pseudo-Boolean constraint

\[-3x_1 + 4x_2 - 7x_3 + x_4 \leq -5\]

- variables $x_i$ take their value in \{0, 1\}
- $\overline{x_1} = 1 - x_1$
- coefficients and degree are integral constants

Pseudo-Boolean decision problem: NP-complete

\[
\begin{align*}
(a_1) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & \quad 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\
(b) & \quad x_1 + x_3 + x_4 \geq 2 \\
(c) & \quad x_1 + \overline{x_2} + x_5 \geq 1
\end{align*}
\]

Plus an objective function: Optimization problem, NP-hard

\[\text{min} : 4x_2 + 2x_3 + x_5\]
Solving Pseudo Boolean Optimization problems with a SAT solver

- Pseudo-Boolean constraints express a boolean formula → that formula can be expressed by a CNF
- Handling those constraints natively in a CDCL solver isn’t hard either (Satire, Satzoo, Minisat, ...) : simplifies the mapping from domain constraints and model constraints, explanations.
- One can easily use a SAT solver to solve an optimization problem using either linear or binary search on the objective function.
Optimization using strengthening (linear search)

**input**: A set of clauses, cardinalities and pseudo-boolean constraints `setOfConstraints` and an objective function `objFct` to minimize

**output**: a model of `setOfConstraints`, or `UNSAT` if the problem is unsatisfiable.

```plaintext
answer ← isSatisfiable (setOfConstraints);
if answer is UNSAT then
  return UNSAT
end
repeat
  model ← answer;
  answer ← isSatisfiable (setOfConstraints ∪
  {objFct < objFct (model)});
until (answer is UNSAT);
return model;
```
Optimization algorithm

Formula:

\[
\begin{cases}
(a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 5 \\
(b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\]

Objective function

\[\min : \quad 4x_2 + 2x_3 + x_5\]
Optimization algorithm

Formula:

\[
\begin{align*}
(a_1) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + 5x_5 \geq 5 \\
(b) & \quad x_1 + x_3 + x_4 \geq 2
\end{align*}
\]

Model

\[x_1, x_2, x_3, x_4, x_5\]

Objective function

\[\text{min : } 4x_2 + 2x_3 + x_5\]
Optimization algorithm

Formula:

\[
\begin{cases}
(a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & 5\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 + \overline{x}_5 \geq 5 \\
(b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\]

Model

\[x_1, x_2, \overline{x}_3, \overline{x}_4, \overline{x}_5\]

Objective function

\[
\min : 4x_2 + 2x_3 + x_5
\]

Objective function value

\[
< 5
\]
Optimization algorithm

Formula:

\[
\begin{cases}
(a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 5 \\
(b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\]

Objective function

\[
\min : 4x_2 + 2x_3 + x_5 < 5
\]
Optimization algorithm

Formula:

\[
\begin{align*}
(a_1) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & \quad 5x_1 + 3x_2 + 2\bar{x}_3 + 2\bar{x}_4 + \bar{x}_5 \geq 5 \\
(b) & \quad x_1 + x_3 + x_4 \geq 2
\end{align*}
\]

Model

\[x_1, \bar{x}_2, x_3, \bar{x}_4, x_5\]

Objective function

\[\min : \quad 4x_2 + 2x_3 + x_5 < 5\]
Optimization algorithm

Formula:

\[
\begin{cases}
(a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & 5\bar{x}_1 + 3\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + \bar{x}_5 \geq 5 \\
(b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\]

Model

\[
x_1, \bar{x}_2, x_3, \bar{x}_4, x_5
\]

Objective function

\[
\text{min : } 4x_2 + 2x_3 + x_5
\]

Objective function value

\[
< 3 < 5
\]
Optimization algorithm

Formula:

\[
\begin{cases}
(a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & 5\bar{x}_1 + 3\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + \bar{x}_5 \geq 5 \\
(b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\]

Objective function

\[
\min : 4x_2 + 2x_3 + x_5 \quad < \quad 3
\]
Optimization algorithm

Formula:

\begin{align*}
\begin{cases}
(a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & 5\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 + \overline{x}_5 \geq 5 \\
(b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\end{align*}

Model

\[ x_1, \overline{x}_2, \overline{x}_3, x_4, x_5 \]

Objective function

\[ \min : \quad 4x_2 + 2x_3 + x_5 \quad < \quad 3 \]
Optimization algorithm

Formula :

\[
\begin{align*}
(a_1) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & \quad 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\
(b) & \quad x_1 + x_3 + x_4 \geq 2
\end{align*}
\]

Model

\[x_1, \overline{x_2}, \overline{x_3}, x_4, x_5\]

Objective function

min : \[4x_2 + 2x_3 + x_5\]

Objective function value

< \[1 < 3\]
Optimization algorithm

Formula:

\[
\begin{align*}
  (a_1) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
  (a_2) & \quad 5\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 + \overline{x}_5 \geq 5 \\
  (b) & \quad x_1 + x_3 + x_4 \geq 2
\end{align*}
\]

Objective function

\[
\min : \quad 4x_2 + 2x_3 + x_5 \quad < \quad 1
\]
Optimization algorithm

Formula:

\[
\begin{cases}
  (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
  (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\
  (b) & x_1 + x_3 + x_4 \geq 2
\end{cases}
\]

Objective function

\[
\text{min} : \quad 4x_2 + 2x_3 + x_5 < 1
\]
Optimization algorithm

Formula:

\[
\begin{align*}
(a_1) & \quad 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\
(a_2) & \quad 5\bar{x}_1 + 3\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + \bar{x}_5 \geq 5 \\
(b) & \quad x_1 + x_3 + x_4 \geq 2
\end{align*}
\]

Objective function

\[
\min : \quad 4x_2 + 2x_3 + x_5
\]

The objective function value 1 is optimal for the formula. \(x_1, \bar{x}_2, \bar{x}_3, x_4, x_5\) is an optimal solution.
Let $C$ be an inconsistent set of clauses.

- $C' \subseteq C$ is an unsat core of $C$ iff $C'$ is inconsistent.

- $C' \subseteq C$ is a MUS of $C$ iff $C'$ is an unsat core of $C$ and no subset of $C'$ is an unsat core of $C$. 
Let $C$ be an inconsistent set of clauses.

$C' \subseteq C$ is an unsat core of $C$ iff $C'$ is inconsistent.

$C' \subseteq C$ is a MUS of $C$ iff $C'$ is an unsat core of $C$ and no subset of $C'$ is an unsat core of $C$.

Computing a MUS (set of clauses) is equivalent to computing the set of literals $L$ such that:

1. $L$ satisfies $\{k_i \lor C_i | C_i \in C\}$
2. $L \cap K$ is subset minimal
Some competitive events are organized for those problems:

- Pseudo Boolean since 2005
- MAX-SAT since 2006
- MUS this year

As such, a common input format exists, together with a bunch of solvers.
Generalized use of selector variables

The minisat+ syndrom : is a SAT solver sufficient for all our needs?

Selector variable principle: satisfying the selector variable should satisfy the selected constraint.

- **Clause**: simply add a new variable
  \[ \bigvee l_i \Rightarrow s \lor \bigvee l_i \]

- **Cardinality**: add a new weighted variable
  \[ \sum l_i \geq d \Rightarrow d \times s + \sum l_i \geq d \]
  The new constraints is PB, no longer a cardinality!

- **Pseudo**: add a new weighted variable
  \[ \sum w_i \times l_i \geq d \Rightarrow d \times s + \sum w_i \times l_i \geq d \]
  if the weights are positive, else use
  \[ (d + \sum_{w_i < 0} |w_i|) \times s + \sum w_i \times l_i \geq d \]
Once cardinality constraints, pseudo boolean constraints and objective functions are managed in a solver, one can easily build a weighted partial Max SAT solver.

- Add a selector variable $s_i$ per soft clause $C_i : s_i \lor C_i$
- Objective function: minimize $\sum s_i$
- Partial MAX SAT: no selector variables for hard clauses
- Weighted MAXSAT: use a weighted sum instead of a sum.

Special case: do not add new variables for unit weighted clauses $w_k/l_k$

Ignore the constraint and add simply $w_k \times \overline{l_k}$ to the objective function.
Recent advances in practical Max Sat solving rely on unsat core computation [Fu and Malik 2006] :

- Compute one unsat core $C'$ of the formula $C$
- Relax it by replacing $C'$ by $\{ r_i \lor C_i | C_i \in C' \}$
- Add the constraint $\sum r_i \leq 1$ to $C$
- Repeat until the formula is satisfiable
- If $\text{MinUnsat}(C) = k$, requires $k$ loops.

Many improvement since then (PM1, PM2, MsUncore, etc) : works for Weighted Max Sat, reduction of the number of relaxation variables, etc.
Selector variables + assumptions = explanation (MUS)

- Assumptions available from the beginning in Minisat 1.12 (incremental SAT)
- Add a new selector variable per constraint
- Check for satisfiability assuming that the selector variables are falsified
- If UNSAT, analyze the final root conflict to keep only selector variables involved in the inconsistency
- Apply a minimization algorithm afterward to compute a minimal explanation

Advantages:
- No changes needed in the SAT solver internals
- Works for any kind of constraints!
- See in action during the MUS track of the SAT 2011 competition!
Outline

SAT in theory

SAT in practice: the international SAT competition

SAT solvers for solving “real” problems: the CDCL architecture

Nearby SAT: PBO, MaxSAT, MUS

Using boolean constraints to solve Software Dependency problems

Conclusion

Depending on time: a quick overview of Sat4j
Current softwares are composite!

- Linux distributions: made of packages
- Eclipse application: made of plugins
- Any complex software: made of libraries
- There are requirements between the diverse components
Dependency Management Problem

$P$ a set of packages

$\text{depends}$ requirement constraints

$\text{depends}: P \rightarrow 2^P$

$\text{conflicts}$ impossible configurations

$\text{conflicts}: P \rightarrow 2^P$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, \text{depends}, \text{conflicts})$ iff

$\forall q \in Q, (\forall \text{dep} \in \text{depends}(q), \text{dep} \cap Q \neq \emptyset) \land (\text{conflicts}(q) \cap Q = \emptyset)$. 
Dependency Management Problem

- $P$ a set of packages
- $\text{depends}$ requirement constraints

\[
\text{depends} : P \rightarrow 2^P
\]

- $\text{conflicts}$ impossible configurations

\[
\text{conflicts} : P \rightarrow 2^P
\]

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, \text{depends}, \text{conflicts})$ iff

\[
\forall q \in Q, (\forall \text{dep} \in \text{depends}(q), \text{dep} \cap Q \neq \emptyset) \land (\text{conflicts}(q) \cap Q = \emptyset).
\]

What is the complexity of finding if a $Q$ containing a specific package exists?
Just as hard as SAT: NP-complete!

See how to decide satisfiability of \((\neg a \lor b \lor c) \land (\neg a \lor \neg b \lor c) \land a \land \neg c\)

```plaintext
package: a
version: 1
conflicts: a = 2

package: clause
version: 1
depends: a = 2 \mid b = 1 \mid c = 1

package: a
version: 2
conflicts: a = 1

package: clause
version: 2
depends: a = 2 \mid b = 2 \mid c = 1

package: b
version: 1
conflicts: b = 2

package: clause
version: 3
depends: a = 1

package: b
version: 2
conflicts: b = 1

package: clause
version: 4
depends: c = 2

package: c
version: 1
conflicts: c = 2

package: clause
version: 1
depends: clause = 1, clause = 2, clause = 3, clause = 4

package: c
version: 2
conflicts: c = 1

request: satisfiability
install: formula
```
Dependencies expressed by clauses

- Dependencies can easily be translated into clauses:

  package: a
  version: 1
  depends: b = 2 | b = 1, c = 1

  \[ a_1 \rightarrow (b_2 \lor b_1) \land c_1 \]

  \[ \neg a_1 \lor b_2 \lor b_1, \neg a_1 \lor c_1 \]

- Conflict can easily be translated into binary clauses:

  package: a
  version: 1
  conflicts: b = 2, d = 1

  \[ \neg a_1 \lor \neg b_2, \neg a_1 \lor \neg d_1 \]
From decision to optimization

- NP-complete, so we can use a SAT solver to solve it
- Finding a solution is usually not sufficient!
  - Minimizing the number of installed packages
  - Minimizing the size of installed packages
  - Keeping up to date versions of packages
  - Preferring most recent packages to older ones
  - ...
- In practice an aggregation of various criteria
- Need a more expressive representation language than plain CNF!
Representing optimization criteria with MaxSat?

\[ \alpha \equiv \bigwedge_{p_v \in P} (p_v \rightarrow (\bigwedge_{\text{dep} \in \text{depends}(p_v)} \text{dep}), \infty) \land \bigwedge_{\text{conf} \in \text{conflicts}(p_v)} (p_v \rightarrow \neg \text{conf}, \infty, \land (q, \infty)) \]

denote the formula to satisfy for installing \( q \).

Minimizing the number of installed packages (Partial MaxSat):

\[ \phi \equiv (\bigwedge_{p_v \in P, p_v \neq q} (\neg p_v, k)) \] (1)

Minimizing the size of installed packages (Weighted Partial MaxSat):

\[ \phi \equiv (\bigwedge_{p_v \in P, p_v \neq q} (\neg p_v, \text{size}(p_v))) \] (2)

Those problems are really \textbf{Binate Covering Problems} (CNF + objective function).
Representing optimization criteria using pseudo-boolean optimization

- We can now rewrite the previous optimization criteria in a simpler manner:
  - Minimizing the number of installed packages:
    \[
    \min : \sum_{p_v \in P, p_v \neq q} p_v
    \]
  - Minimizing the size of installed packages:
    \[
    \min : \sum_{p_v \in P, p_v \neq q} \text{size}(p_v) \times p_v
    \]
  - We can express easily that only one version of package \textit{libnss} can be installed:
    \[
    \text{libnss}_1 + \text{libnss}_2 + \text{libnss}_3 + \text{libnss}_4 + \text{libnss}_5 \leq 1
    \]
The full encoding

- Contains Eclipse specific features (patches, root packages, etc).
- Has a specific optimization function tailored for two years thanks to user feedback.
- Up and running since Eclipse 3.4 (June 2008).
- For both:
  - Building Eclipse-based products
  - Updating and Eclipse installation
Outline

SAT in theory

SAT in practice: the international SAT competition

SAT solvers for solving “real” problems: the CDCL architecture

Nearby SAT: PBO, MaxSAT, MUS

Using boolean constraints to solve Software Dependency problems

Conclusion

Depending on time: a quick overview of Sat4j
Conclusion

- Many efficient SAT solvers available
- Can be used to solve either decision or optimization problems
- Success stories for decision problems (Intel Core i7, Windows 7)
- Optimization problems receiving more and more attention (MaxSat, PBO, WBO, SMT OPT, etc)
- Efficient optimization on “real” test cases often rely on raw SAT solvers (encoding Minisat+, unsat core for MaxSat)
- Reusing SAT engines is key for solving efficiently those problems
- because SAT engines are still more and more efficient
- Not many SAT library fully featured to offer to the end user state-of-the-art boolean constraints solver and optimizer.
Outline

SAT in theory

SAT in practice: the international SAT competition

SAT solvers for solving “real” problems: the CDCL architecture

Nearby SAT: PBO, MaxSAT, MUS

Using boolean constraints to solve Software Dependency problems

Conclusion

Depending on time: a quick overview of Sat4j
Birth of SAT4J, Java implementation of Minisat.
Open Source (licensed under GNU LGPL).
Sat4j, from ADS to p2cudf

- Joint project with INESC → PB evaluation + Sat4j PB
- First CSP competition → release of Sat4j CSP
First MaxSAT evaluation
Birth of Sat4j MaxSAT
Sat4j, from ADS to p2cudf

Integration within Eclipse [IWOCE09]
Sat4j relicensed under both EPL and GNU LGPL
Sat4j, from ADS to p2cudf

- ADS
- JSAT
- OpenSAT
- SAT4J
- PB
- MAXSAT
- Eclipse
- p2cudf

DPLL ERA
CDCL ERA


Chaff
Minisat

- Application to Linux dependencies: p2cudf [LoCoCo 2010]
- Licensed under EPL
Sat4j, from ADS to p2cufd
Initial aim of the project

- Providing 100% Java SAT technology
- With an Open Source software
- Flexible enough to experiment our ideas
- Efficient enough to solve real problems
- Designed to be used in academia or production software
A flexible framework for solving propositional problems

- explanation
- optimization
- constraints
- PB constraints
  - cardinalities
  - clauses
- inference
- cutting planes
- resolution

generic CDCL
SAT4J today

- SAT4J PB (Res, CP) are not very efficient, but correct (arbitrary precision arithmetic).
- SAT4J SAT solvers can be found in various software from academia (Alloy 4, Forge, ....) to commercial applications (GNA.sim).
- SAT4J PB Res solves Eclipse plugin dependencies since June 2008 (Eclipse 3.4, Ganymede)
  - SAT4J ships with every product based on the Eclipse platform (more than 13 millions downloads per year from Eclipse.org since June 2008)
  - SAT4J helps to build Eclipse products daily (e.g. nightly builds on Eclipse.org, IBM, SAP, etc)
  - SAT4J helps to update Eclipse products worldwide daily
Summary of SAT4J functionalities

Decision Problems

- Sat
- Pseudo-boolean Problems
  - Clauses/Cardinalities
  - Clauses/Cardinalities/PB Constraints
  - Resolution
  - Cutting Planes
- Sat4j-Core
- Sat4j-PB-Res
- Sat4j-PB-CP

Optimization problems

- Partial Weighted MaxSat
- PB Optimization/WBO