Tutorial on SAT

Armin Biere

Johannes Kepler Universität Linz

CP’17   ILCP’17   SAT’17

Melbourne, Australia

August 29, 2017
propositional logic:

- **variables**
  - tie
  - shirt

- **negation** \( \neg \)
  - (not)

- **disjunction** \( \lor \)
  - (or)

- **conjunction** \( \land \)
  - (and)

**clauses (conditions / constraints)**

1. clearly one should not wear a tie without a shirt
   \[
   \neg \text{tie} \lor \text{shirt}
   \]

2. not wearing a tie nor a shirt is impolite
   \[
   \text{tie} \lor \text{shirt}
   \]

3. wearing a tie and a shirt is overkill
   \[
   \neg (\text{tie} \land \text{shirt}) \equiv \neg \text{tie} \lor \neg \text{shirt}
   \]

Is this formula in conjunctive normal form (CNF) **satisfiable**?

\[
(\neg \text{tie} \lor \text{shirt}) \land (\text{tie} \lor \text{shirt}) \land (\neg \text{tie} \lor \neg \text{shirt})
\]
SAT-Race 2010 Award
“Plingeling”
by Armin Biere
is awarded the title of
Best Parallel Solver

SAT-Race 2010 Award
“Lingeling”
by Armin Biere
is awarded the title of
Second Prize Winner

Tenth International Conference
On Theory and Applications of Satisfiability Testing
Competition 2007
Gold medal

Competition 2009
3rd medal

Competition 2010
2nd medal

Tenth International Conference
On Theory and Applications of Satisfiability Testing
Competition 2002
Winner

Competition 2009
Silver medal

Tenth International Conference
On Theory and Applications of Satisfiability Testing
Competition 2015
Special Prize: Most Innovative Solver
Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and ... for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. Thanks also to Stanford’s Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

* * * * *

Wow—Section 7.2.2.2 has turned out to be the longest section, by far, in The Art of Computer Programming. The SAT problem is evidently a “killer app,” because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of TAOCP doesn’t allow for a Section 7.2.2.2.1.)

I’ve tried to ameliorate the reader’s navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided.
What is Practical SAT Solving?

- encoding
- reencoding
- simplifying
- inprocessing
- search
- CDCL
Equivalence Checking If-Then-Else Chains

original C code

```c
if(!a && !b) h();
else if(!a) g();
else f();
```

optimized C code

```c
if(a) f();
else if(b) g();
else h();
```

How to check that these two versions are equivalent?
Compilation

\[ \text{original} \equiv \begin{cases} \text{if } \neg a \land \neg b \text{ then } h \text{ else if } \neg a \text{ then } g \text{ else } f \\ (\neg a \land \neg b) \land h \lor (\neg a \land \neg b) \land (\neg a \land g \lor a \land f) \end{cases} \]

\[ \text{optimized} \equiv \begin{cases} \text{if } a \text{ then } f \text{ else if } b \text{ then } g \text{ else } h \\ a \land f \lor \neg a \land (b \land g \lor \neg b \land h) \end{cases} \]

\[ (\neg a \land \neg b) \land h \lor (\neg a \land \neg b) \land (a \land f) \not\equiv a \land f \lor \neg a \land (b \land g \lor \neg b \land h) \]

satisfying assignment gives counter-example to equivalence
Tseitin Transformation: Circuit to CNF

\[ o \land (x \rightarrow a) \land (x \rightarrow c) \land (x \leftarrow a \land c) \land \ldots \]

\[ o \land (\overline{x} \lor a) \land (\overline{x} \lor c) \land (x \lor \overline{a} \lor \overline{c}) \land \ldots \]
Negation: \[ x \leftrightarrow \bar{y} \iff (x \rightarrow \bar{y}) \land (\bar{y} \rightarrow x) \]
\[ \iff (\bar{x} \lor \bar{y}) \land (y \lor x) \]

Disjunction: \[ x \leftrightarrow (y \lor z) \iff (y \rightarrow x) \land (z \rightarrow x) \land (x \rightarrow (y \lor z)) \]
\[ \iff (\bar{y} \lor x) \land (\bar{z} \lor x) \land (\bar{x} \lor y \lor z) \]

Conjunction: \[ x \leftrightarrow (y \land z) \iff (x \rightarrow y) \land (x \rightarrow z) \land ((y \land z) \rightarrow x) \]
\[ \iff (\bar{x} \lor y) \land (\bar{x} \lor z) \land ((y \land z) \lor x) \]
\[ \iff (\bar{x} \lor y) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z} \lor x) \]

Equivalence: \[ x \leftrightarrow (y \leftrightarrow z) \iff (x \rightarrow (y \leftrightarrow z)) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \iff (x \rightarrow ((y \rightarrow z) \land (z \rightarrow y))) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \iff (x \rightarrow (y \rightarrow z)) \land (x \rightarrow (z \rightarrow y)) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \iff (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \iff (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land ((y \land z) \lor (\bar{y} \land \bar{z})) \rightarrow x) \]
\[ \iff (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land ((y \land z) \rightarrow x) \land ((\bar{y} \land \bar{z}) \rightarrow x) \]
\[ \iff (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land (\bar{y} \lor \bar{z} \lor x) \land (y \lor z \lor x) \]
Bit-Blasting of Bit-Vector Addition

addition of 4-bit numbers \(x, y\) with result \(s\) also 4-bit: \(s = x + y\)

\[
[s_3, s_2, s_1, s_0]_4 = [x_3, x_2, x_1, x_0]_4 + [y_3, y_2, y_1, y_0]_4
\]

\[
[s_3, c_2]_2 = \text{FullAdder}(x_3, y_3, c_2)
\]
\[
[s_2, c_2]_2 = \text{FullAdder}(x_2, y_2, c_1)
\]
\[
[s_1, c_1]_2 = \text{FullAdder}(x_1, y_1, c_0)
\]
\[
[s_0, c_0]_2 = \text{FullAdder}(x_0, y_0, \text{false})
\]

where

\[
[s, o]_2 = \text{FullAdder}(x, y, i) \quad \text{with}
\]
\[
s = x \text{xor} y \text{xor} i
\]
\[
o = (x \land y) \lor (x \land i) \lor (y \land i) = ((x + y + i) \geq 2)
\]
Intermediate Representations

- encoding directly into CNF is hard, so we use intermediate levels:
  1. application level
  2. bit-precise semantics world-level operations (bit-vectors)
  3. bit-level representations such as And-Inverter Graphs (AIGs)
  4. conjunctive normal form (CNF)
- encoding “logical” constraints is another story
XOR as AIG

\[
x \text{ xor } y \equiv (\overline{x} \land y) \lor (x \land \overline{y}) \equiv \overline{(\overline{x} \land y)} \land (x \land \overline{y})
\]

negation/sign are edge attributes
not part of node
4-bit adder

8-bit adder
bit-vector of length 16 shifted by bit-vector of length 4
Tseitin’s construction suitable for most kinds of “model constraints”
- assuming simple operational semantics: encode an interpreter
- small domains: one-hot encoding  large domains: binary encoding

harder to encode properties or additional constraints
- temporal logic / fix-points
- environment constraints

example for fix-points / recursive equations:  
\[ x = (a \lor y), \quad y = (b \lor x) \]
- has unique least fix-point  \[ x = y = (a \lor b) \]
- and unique largest fix-point  \[ x = y = true \] but unfortunately …
- … only largest fix-point can be (directly) encoded in SAT
  otherwise need stable models / logical programming / ASP
Example of Logical Constraints: Cardinality Constraints

- given a set of literals \( \{l_1, \ldots l_n\} \)
  - constraint the **number** of literals assigned to **true**
  - \( l_1 + \cdots + l_n \geq k \) or \( l_1 + \cdots + l_n \leq k \) or \( l_1 + \cdots + l_n = k \)
  - combined make up exactly all fully symmetric boolean functions

- multiple encodings of cardinality constraints
  - naïve encoding exponential: **at-most-one** quadratic, **at-most-two** cubic, etc.
  - quadratic \( O(k \cdot n) \) encoding goes back to Shannon
  - linear \( O(n) \) parallel counter encoding [Sinz'05]

- many variants even for **at-most-one** constraints
  - for an \( O(n \cdot \log n) \) encoding see Prestwich’s chapter in our Handbook of SAT

- **Pseudo-Boolean** constraints (PB) or 0/1 ILP constraints have many encodings too

\[
2 \cdot \overline{a} + b + c + \overline{d} + 2 \cdot e \geq 3
\]

actually used to handle MaxSAT in SAT4J for configuration in Eclipse
BDD-Based Encoding of Cardinality Constraints

\[ 2 \leq l_1 + \cdots l_9 \leq 3 \]

If-Then-Else gates (MUX) with “then” edge downward, dashed “else” edge to the right
Tseitin Encoding of If-Then-Else Gate

\[
x \leftrightarrow (c \ ? \ t : e) \iff (x \rightarrow (c \rightarrow t)) \land (x \rightarrow (\bar{c} \rightarrow e)) \land (\bar{x} \rightarrow (c \rightarrow \bar{t})) \land (\bar{x} \rightarrow (\bar{c} \rightarrow \bar{e}))
\]

\[
\iff (\bar{x} \lor \bar{c} \lor t) \land (\bar{x} \lor c \lor e) \land (x \lor \bar{c} \lor \bar{t}) \land (x \lor c \lor \bar{e})
\]

minimal but not arc consistent:

- if \( t \) and \( e \) have the same value then \( x \) needs to have that too
- possible additional clauses
  \[
  (\bar{t} \land \bar{e} \rightarrow \bar{x}) \equiv (t \lor e \lor \bar{x}) \quad \quad (t \land e \rightarrow x) \equiv (\bar{t} \lor \bar{e} \lor x)
  \]
- but can be learned or derived through preprocessing (ternary resolution)
  keeping those clauses redundant is better in practice
DP / DPLL

- dates back to the 50’ies:
  - 1\textsuperscript{st} version DP is resolution based \Rightarrow \text{preprocessing}
  - 2\textsuperscript{nd} version D(P)LL splits space for time \Rightarrow \text{CDCL}

- ideas:
  - 1\textsuperscript{st} version: eliminate the two cases of assigning a variable in space or
  - 2\textsuperscript{nd} version: case analysis in time, e.g. try $x = 0, 1$ in turn and recurse

- most successful SAT solvers are based on variant (CDCL) of the second version
  - works for very large instances

- recent ($\leq 20$ years) optimizations:
  - backjumping, learning, UIPs, dynamic splitting heuristics, fast data structures
  - (we will have a look at each of them)
DP Procedure

forever

if $F = \top$ return satisfiable

if $\bot \in F$ return unsatisfiable

pick remaining variable $x$

add all resolvents on $x$

remove all clauses with $x$ and $\neg x$

$\Rightarrow$ bounded variable elimination in SatELite preprocessor
Bounded Variable Elimination

[EénBiere-SAT’05]

Replace

\[
\begin{align*}
(\bar{x} \lor a)_1 & \quad (\bar{x} \lor c)_4 & \quad (a \lor \bar{a} \lor \bar{b})_{13} & \quad (a \lor d)_{15} & \quad (c \lor d)_{45} \\
(\bar{x} \lor b)_2 & \quad (x \lor d)_5 & \quad (b \lor \bar{a} \lor \bar{b})_{23} & \quad (b \lor d)_{25} & \quad (c \lor \bar{a} \lor \bar{b})_{34}
\end{align*}
\]

- number of clauses not increasing
- strengthen and remove subsumed clauses too
- most important and most effective preprocessing we have

Bounded Variable Addition

[MantheyHeuleBiere-HVC’12]

Replace

\[
\begin{align*}
(a \lor d) & \quad (a \lor e) & \quad (\bar{x} \lor a) & \quad (\bar{x} \lor b) & \quad (\bar{x} \lor c) \\
(b \lor d) & \quad (b \lor e) & \quad (x \lor d) & \quad (x \lor e) & \\
(c \lor d) & \quad (c \lor e) &
\end{align*}
\]

- number of clauses has to decrease strictly
- reencodes for instance naive at-most-one constraint encodings
D(P)LL Procedure

\[ DPLL(F) \]

\[ F := BCP(F) \]

if \( F = \top \) return satisfiable

if \( \bot \in F \) return unsatisfiable

pick remaining variable \( x \) and literal \( l \in \{ x, \neg x \} \)

if \( DPLL(F \land \{l\}) \) returns satisfiable return satisfiable

return \( DPLL(F \land \{\neg l\}) \)

\( \Rightarrow \) CDCL
DPLL Example

\[
\begin{align*}
& a = 1 \\
& b = 1 \\
& c = 0
\end{align*}
\]

BCP

\[
\begin{align*}
& \neg a \lor \neg b \lor \neg c \\
& \neg a \\
& \neg a \lor \neg b \\
& a \lor \neg b \\
& a \lor \neg b \lor c
\end{align*}
\]
Simple Data Structures in DPLL Implementation
BCP Example

<table>
<thead>
<tr>
<th>Variables</th>
<th>Assignment</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>1</td>
<td>-1 2</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
<td>-2 3</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>4</td>
<td>-4 5</td>
</tr>
<tr>
<td>X</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example cont.

Decide

<table>
<thead>
<tr>
<th>Variables</th>
<th>Assignment</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 1</td>
<td></td>
<td>-1 2</td>
</tr>
<tr>
<td>X 2</td>
<td></td>
<td>-2 3</td>
</tr>
<tr>
<td>X 3</td>
<td></td>
<td>-4 5</td>
</tr>
<tr>
<td>X 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

decision level

0

0

Trail

Control
Example cont.

Assign

decision level

Control

Trail

Variables

Assignment

Clauses

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example cont.

BCP

decision level  Control  Trail

Assignment

Variables

Clauses

1 1
1 2
1 3
X 4
X 5
Example cont.

```
<table>
<thead>
<tr>
<th>Assignment</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 •</td>
<td>-1 2</td>
</tr>
<tr>
<td>1 2 •</td>
<td>-2 3</td>
</tr>
<tr>
<td>1 3 •</td>
<td>-4 5</td>
</tr>
<tr>
<td>X 4 •</td>
<td></td>
</tr>
<tr>
<td>X 5 •</td>
<td></td>
</tr>
</tbody>
</table>
```

```
Decision level
2

Control
3
0
0

Trail
3
2
1
```

```
Decide
```

```
X
X
```

```
```
Example cont.

Assign

decision level

Control

Trail

Variables

Assignment

Clauses

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>3</td>
</tr>
<tr>
<td>-4</td>
<td>5</td>
</tr>
</tbody>
</table>
Example cont.

BCP

Decision level

Control

Trail

Variables

Assignment

Clauses

-1 2

-2 3

-4 5
Conflict Driven Clause Learning (CDCL)

- first implemented in the context of GRASP SAT solver [MarqueSilvaSakallah’96]
  - name given later to distinguish it from DPLL
  - not recursive anymore
- essential for SMT
- learning clauses as no-goods
- notion of implication graph
- (first) unique implication points
Conflict Driven Clause Learning (CDCL)

\[
\begin{align*}
\text{decision} & : a \\
\text{BCP} & : \neg c \\
 a = 1 & \\
b = 1 & \\
c = 0 & \\
\text{clauses} & : \\
\neg a \lor \neg b \lor \neg c & \\
\neg a \lor \neg b \lor c & \\
\neg a \lor b \lor \neg c & \\
\neg a \lor b \lor c & \\
a \lor \neg b \lor \neg c & \\
a \lor \neg b \lor c & \\
a \lor b \lor \neg c & \\
a \lor b \lor c & \\
\text{learn} & : \neg a \lor \neg b
\end{align*}
\]
Conflict Driven Clause Learning (CDCL)

- Decision: $a$
- $a = 1$
- $b = 0$
- $c = 0$

Clauses:

- $\neg a \lor \neg b \lor c$
- $\neg a \lor \neg b \lor c$
- $a \lor \neg b \lor \neg c$
- $a \lor b \lor \neg c$
- $a \lor b \lor c$
- $\neg a \lor \neg b$
- $\neg a$

Decision BCP (Backward Conflict Propagation): $\neg b$
Conflict Driven Clause Learning (CDCL)

\[
a = 1
\]
\[
b = 0
\]
\[
c = 0
\]

\[
\neg a \quad \text{BCP}
\]
\[
\neg b \quad \text{BCP}
\]
\[
\neg c \quad \text{decision}
\]

clauses

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

learn

\[
c
\]
Conflict Driven Clause Learning (CDCL)

\( a = 1 \)
\( b = 0 \)
\( c = 0 \)

clauses

\( \neg a \lor \neg b \lor \neg c \)
\( \neg a \lor b \lor \neg c \)
\( \neg a \lor \neg b \lor c \)
\( a \lor b \lor c \)

learn

\( \bot \)

empty clause
Conflict

top−level

unit

a = 1 @ 0

unit

b = 1 @ 0

decision

c = 1 @ 1

d = 1 @ 1

e = 1 @ 1

decision

f = 1 @ 2

g = 1 @ 2

h = 1 @ 2

i = 1 @ 2

decision

k = 1 @ 3

l = 1 @ 3

decision

r = 1 @ 4

s = 1 @ 4

t = 1 @ 4

y = 1 @ 4

x = 1 @ 4

z = 1 @ 4

κ

conflict
Antecedents / Reasons

\[
d \land g \land s \rightarrow t \equiv (\overline{d} \lor \overline{g} \lor \overline{s} \lor t)
\]
Conflicting Clauses

\[ \neg (y \land z) \equiv (\bar{y} \lor \bar{z}) \]
Resolving Antecedents 1st Time

\[(\overline{h} \lor \overline{i} \lor \overline{t} \lor y) \quad (\overline{y} \lor z)\]
Resolving Antecedents 1st Time

\[ \begin{align*}
\text{top-level} & \quad \text{unit} \quad a = 1 @ 0 \quad \text{unit} \quad b = 1 @ 0 \\
\text{decision} & \quad c = 1 @ 1 \quad d = 1 @ 1 \quad e = 1 @ 1 \\
\text{decision} & \quad f = 1 @ 2 \quad g = 1 @ 2 \\
\text{decision} & \quad k = 1 @ 3 \quad l = 1 @ 3 \\
\text{decision} & \quad r = 1 @ 4 \quad s = 1 @ 4 \\
\text{decision} & \quad x = 1 @ 4 \\
\end{align*} \]

\[ h = 1 @ 2 \quad i = 1 @ 2 \quad t = 1 @ 4 \quad y = 1 @ 4 \quad z = 1 @ 4 \]

\[ \kappa \text{ conflict} \]

\[ (\bar{h} \lor \bar{i} \lor \bar{t} \lor \bar{y}) \quad (\bar{y} \lor \bar{z}) \]

\[ (\bar{h} \lor \bar{i} \lor \bar{t} \lor \bar{z}) \]
Resolvents = Cuts = Potential Learned Clauses

\[
\frac{(\overline{h} \lor \overline{i} \lor \overline{t} \lor y)}{y \lor z} \quad \frac{(\overline{y} \lor \overline{z})}{(\overline{h} \lor \overline{i} \lor \overline{t} \lor \overline{z})}
\]
Potential Learned Clause After 1 Resolution

\[(\overline{h} \lor \overline{i} \lor \overline{i} \lor \overline{z})\]
Resolving Antecedents 2nd Time

\[
\begin{align*}
\text{top-level} & \quad \text{unit} \quad a = 1 \, @ \, 0 & \quad \text{unit} \quad b = 1 \, @ \, 0 \\
\text{decision} \quad c = 1 \, @ \, 1 & \quad d = 1 \, @ \, 1 & \quad e = 1 \, @ \, 1 \\
\text{decision} \quad f = 1 \, @ \, 2 & \quad g = 1 \, @ \, 2 & \quad h = 1 \, @ \, 2 & \quad i = 1 \, @ \, 2 \\
\text{decision} \quad k = 1 \, @ \, 3 & \quad l = 1 \, @ \, 3 \\
\text{decision} \quad r = 1 \, @ \, 4 & \quad s = 1 \, @ \, 4 & \quad t = 1 \, @ \, 4 & \quad y = 1 \, @ \, 4 \\
\text{decision} \quad x = 1 \, @ \, 4 & \quad z = 1 \, @ \, 4 & \quad \kappa & \text{conflict} \\
\end{align*}
\]

\[
(d \lor g \lor s \lor t) \quad (h \lor \bar{i} \lor \bar{i} \lor \bar{z})
\]

\[
(d \lor g \lor s \lor h \lor \bar{i} \lor \bar{z})
\]
Resolving Antecedents 4th Time

\[
\begin{align*}
&\text{top-level} \quad \text{unit} \quad a = 1 \atop 0 \quad \text{unit} \quad b = 1 \atop 0 \\
\text{decision} \quad c = 1 \atop 1 \quad d = 1 \atop 1 \quad e = 1 \atop 1 \\
\text{decision} \quad f = 1 \atop 2 \quad g = 1 \atop 2 \quad h = 1 \atop 2 \quad i = 1 \atop 2 \\
\text{decision} \quad k = 1 \atop 3 \quad l = 1 \atop 3 \\
\text{decision} \quad r = 1 \atop 4 \quad s = 1 \atop 4 \quad t = 1 \atop 4 \quad y = 1 \atop 4 \\
\quad x = 1 \atop 4 \\
\quad z = 1 \atop 4 \\
\quad \kappa = \text{conflict}
\end{align*}
\]

\[
\begin{align*}
&(\bar{s} \lor x) \\
&(\bar{x} \lor \bar{d} \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i})
\end{align*}
\]

self subsuming resolution
1st UIP Clause after 4 Resolutions

$$\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor \overline{i}$$

$$\text{UIP} = \text{unique implication point} \quad \text{dominates conflict on the last level}$$
Resolving Antecedents 5th Time

\[
\begin{align*}
\text{top−level} & \quad \text{unit} \quad a = 1 @ 0 \quad \text{unit} \quad b = 1 @ 0 \\
decision \quad c = 1 @ 1 & \quad d = 1 @ 1 & \quad e = 1 @ 1 \\
decision \quad f = 1 @ 2 & \quad g = 1 @ 2 & \quad h = 1 @ 2 & \quad i = 1 @ 2 \\
decision \quad k = 1 @ 3 & \quad l = 1 @ 3 \\
decision \quad r = 1 @ 4 & \quad s = 1 @ 4 & \quad t = 1 @ 4 & \quad y = 1 @ 4 \\
x = 1 @ 4 & \quad z = 1 @ 4 & \quad p \quad \text{conflict}
\end{align*}
\]

\[
(\bar{l} \lor \bar{r} \lor \bar{s}) \quad (\bar{d} \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i})
\]

\[
(\bar{l} \lor \bar{r} \lor \bar{d} \lor \bar{g} \lor \bar{h} \lor \bar{i})
\]
Decision Learned Clause

\[(d \lor \neg g \lor \neg l \lor \neg r \lor \neg h \lor \neg i)\]
1st UIP Clause after 4 Resolutions

\[ (\bar{d} \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i}) \]
Locally Minimizing 1st UIP Clause

(top-level) (unit) $a = 1 @ 0$ (unit) $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

decision $k = 1 @ 3$ $l = 1 @ 3$

decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

$$(\overline{h} \lor i) \quad (\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor \overline{i})$$

$$(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h})$$

self subsuming resolution
Locally Minimized Learned Clause

\[ (\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h}) \]
Minimizing Locally Minimized Learned Clause Further?

\[(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h})\]

\[a = 1 \land 0\]
\[b = 1 \land 0\]
\[c = 1 \land 1\]
\[d = 1 \land 1\]
\[e = 1 \land 1\]
\[f = 1 \land 2\]
\[g = 1 \land 2\]
\[h = 1 \land 2\]
\[i = 1 \land 2\]
\[k = 1 \land 3\]
\[l = 1 \land 3\]
\[r = 1 \land 4\]
\[s = 1 \land 4\]
\[t = 1 \land 4\]
\[y = 1 \land 4\]
\[x = 1 \land 4\]
\[z = 1 \land 4\]
\[\kappa = \text{conflict}\]
Recursively Minimizing Learned Clause

\[(b) \quad (\overline{b} \lor \overline{d} \lor \overline{g} \lor \overline{s})\]

\[(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h})\]

\[(\overline{e} \lor \overline{g} \lor h)\]

\[(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h})\]

\[(\overline{e} \lor \overline{d} \lor \overline{g} \lor \overline{s})\]
Recursively Minimized Learned Clause

\((\overline{d} \lor \overline{g} \lor \overline{s})\)
Decision Heuristics

- number of variable occurrences in (remaining unsatisfied) clauses (LIS)
  - eagerly satisfy many clauses
  - many variations were studied in the 90ies
  - actually expensive to compute

- dynamic heuristics
  - **focus on variables which were useful recently in deriving learned clauses**
  - can be interpreted as reinforcement learning
  - started with the VSIDS heuristic [MoskewiczMadiganZhaoZhangMalik’01]
  - most solvers rely on the exponential variant in MiniSAT (EVSIDS)
  - recently showed VMTF as effective as VSIDS [Biere-SAT’15] acts as survey

- look-ahead
  - spent more time in selecting good variables (and simplification)
  - related to our Cube & Conquer paper [HeuleKullmanWieringaBiere-HVC’11]
  - “The Science of Brute Force” [Heule & Kullman CACM August 2017]
## Variable Scoring Schemes

\[ s \quad \text{old score} \quad s' \quad \text{new score} \]

<table>
<thead>
<tr>
<th>Variable score ( s' ) after ( i ) conflicts</th>
<th>Bumped</th>
<th>Not-bumped</th>
</tr>
</thead>
</table>
| **STATIC**                                      | \( s \) | \( s \)      | static decision order  
| **INC**                                         | \( s + 1 \) | \( s \)      | increment scores      
| **SUM**                                         | \( s + i \) | \( s \)      | sum of conflict-indices  
| **VSIDS**                                       | \( h_i^{256} \cdot s + 1 \) | \( h_i^{256} \cdot s \) | original implementation in Chaff  
| **NVSIDS**                                      | \( f \cdot s + (1 - f) \) | \( f \cdot s \) | normalized variant of VSIDS  
| **EVSIDS**                                      | \( s + g^i \) | \( s \)      | exponential MiniSAT dual of NVSIDS  
| **ACIDS**                                       | \( (s + i)/2 \) | \( s \)      | average conflict-index decision scheme  
| **VMTF**                                        | \( i \) | \( s \)      | variable move-to-front  
| **VMTF'**                                       | \( b \) | \( s \)      | variable move-to-front variant  

\[
0 < f < 1 \quad g = 1/f \quad h_i^m = 0.5 \quad \text{if } m \text{ divides } i \quad h_i^m = 1 \text{ otherwise} \\
\]

\[ i \quad \text{conflict index} \quad b \quad \text{bumped counter} \]
Backjumping

If $y$ has never been used to derive a conflict, then skip $\overline{y}$ case.

Immediately jump back to the $\overline{x}$ case – assuming $x$ was used.
Basic CDCL Loop

```c
int basic_cdcl_loop () {
    int res = 0;

    while (!res)
        if (unsat) res = 20; // analyze propagated conflict
        else if (!propagate ()) analyze (); // all variables satisfied
        else if (satisfied ()) res = 10; // otherwise pick next decision
        else decide ();

    return res;
}
```
Reducing Learned Clauses

- keeping all learned clauses slows down BCP kind of quadratically
  - so SATO and RelSAT just kept only “short” clauses

- better periodically delete “useless” learned clauses “search cache”
  - keep a certain number of learned clauses
  - if this number is reached MiniSAT reduces (deletes) half of the clauses
  - then maximum number kept learned clauses is increased geometrically

- LBD (glucose level / glue) based prediction for usefulness [AudemardSimon-IJCAI’09]
  - LBD = number of decision-levels in the learned clause
  - allows arithmetic increase of number of kept learned clauses
  - keep clauses with small LBD forever (≤ 2…5)
  - large fixed cache useful for hard satisfiable instances (crypto) [Chanseok Oh]
Restarts

- often it is a good strategy to abandon what you do and restart
  - for satisfiable instances the solver may get stuck in the unsatisfiable part
  - for unsatisfiable instances focusing on one part might miss short proofs
  - restart after the number of conflicts reached a restart limit

- avoid to run into the same dead end
  - by randomization (either on the decision variable or its phase)
  - and/or just keep all the learned clauses

- for completeness dynamically increase restart limit
  - arithmetically, geometrically, Luby, Inner/Outer

- Glucose restarts [AudemardSimon-CP’12]
  - short vs. large window exponential moving average (EMA) over LBD
  - if recent LBD values are larger than long time average then restart
Luby’s Restart Intervals

70 restarts in 104448 conflicts
unsigned
luby (unsigned i)
{
    unsigned k;

    for (k = 1; k < 32; k++)
        if (i == (1 << k) - 1)
            return 1 << (k - 1);

    for (k = 1;; k++)
        if ((1 << (k - 1)) <= i && i < (1 << k) - 1)
            return luby (i - (1 << (k-1)) + 1);
}

limit = 512 * luby (++restarts);
...  // run SAT core loop for 'limit' conflicts
Reluctant Doubling Sequence

[Knuth’12]

\[(u_1, v_1) = (1, 1)\]

\[(u_{n+1}, v_{n+1}) = ((u_n \& -u_n == v_n) \ ? \ (u_n + 1, 1) : (u_n, 2v_n))\]

\[(1, 1), (2, 1), (2, 2), (3, 1), (4, 1), (4, 2), (4, 4), (5, 1), \ldots\]
Phase Saving and Rapid Restarts

- **phase assignment**:
  - assign decision variable to 0 or 1?
  - only thing that matters in *satisfiable* instances

- “phase saving” as in RSat [PipatsrisawatDarwiche’07]
  - pick phase of last assignment (if not forced to, do not toggle assignment)
  - initially use statically computed phase (typically LIS)
  - so can be seen to maintain a **global full assignment**
  - and thus makes CDCL actually a rather “local” search procedure

- rapid restarts
  - varying restart interval with bursts of restarts
  - not only theoretically avoids local minima
  - works nicely together with phase saving

- reusing the trail can reduce the cost of restarts [RamosVanDerTakHeule-JSAT’11]
Restart Scheduling with Exponential Moving Averages

[BiereFröhlich-POS’15]

- LBD
- fast EMA of LBD with $\alpha = 2^{-5}$
- restart
- slow EMA of LBD with $\alpha = 2^{-14}$ (ema-14)
- inprocessing
- CMA of LBD (average)
int basic_cdcl_loop_with_reduce_and_restart () {
    int res = 0;
    while (!res)
        if (unsat) res = 20;
    else if (!propagate ()) analyze (); // analyze propagated conflict
    else if (satisfied ()) res = 10;  // all variables satisfied
    else if (restarting ()) restart ();  // restart by backtracking
    else if (reducing ()) reduce ();   // collect useless learned clauses
    else decide ();                   // otherwise pick next decision
    return res;
}
int Internal::search () {
  int res = 0;
  START (search);
  
while (!res)
         if (unsat) res = 20;
    else if (!propagate ()) analyze (); // analyze propagated conflict
    else if (iterating) iterate (); // report learned unit
    else if (satisfied ()) res = 10; // all variables satisfied
    else if (terminating ()) break; // limit hit or asynchronous abort
    else if (restarting ()) restart (); // restart by backtracking
    else if (reducing ()) reduce (); // collect useless learned clauses
    else if (probing ()) probe (); // failed literal probing
    else if (subsuming ()) subsume (); // subsumption algorithm
    else if (eliminating ()) elim (); // bounded variable elimination
    else if (compactifying ()) compact (); // collect internal variables
    else decide (); // otherwise pick next decision
  STOP (search);
  
return res;
}
Two-Watched Literal Schemes

- original idea from SATO [ZhangStickel'00]
  - invariant: always watch two non-false literals
  - if a watched literal becomes false replace it
  - if no replacement can be found clause is either unit or empty
  - original version used head and tail pointers on Tries

- improved variant from Chaff [MoskewiczMadiganZhaoZhangMalik'01]
  - watch pointers can move arbitrarily
  - no update needed during backtracking
  - one watch is enough to ensure correctness but loses arc consistency
  - reduces visiting clauses by 10x
    - particularly useful for large and many learned clauses

- blocking literals [ChuHarwoodStuckey'09]

- special treatment of short clauses (binary [PilarskiHu’02] or ternary [Ryan’04])

- cache start of search for replacement [Gent-JAIR’13]
Proofs / RUP / DRUP

- original idea for proofs: proof traces / sequence consisting of “learned clauses”
- can be checked clause by clause by unit propagation
- reverse unit implied clauses (RUP) [GoldbergNovikov’03][VanGelder’12]
- deletion information (DRUP): proof trace of added and deleted clauses
- RUP in SAT competition 2007, 2009, 2011, DRUP since 2013 to certify UNSAT

Blocked Clauses
[Kullman-DAM’99] [JärvisaloHeuleBiere-JAR’12]

\[
C \Rightarrow (a \lor l) \quad \text{“blocked” on } l \text{ w.r.t. CNF} \\
F \Rightarrow (\overline{a} \lor b) \land (l \lor c) \land (\overline{l} \lor \overline{a})
\]

- all resolvents of $C$ on $l$ with clauses $D$ in $F$ are tautological
- blocked clauses are “redundant” too
  - adding or removing blocked clauses does not change satisfiability status
  - however it might change the set of models
Resolution Asymmetric Tautologies (RAT)
“Inprocessing Rules” [JärvisaloHeuleBiere-IJCAR’12]

- justify complex preprocessing algorithms in Lingeling
  - examples are adding blocked clauses or variable elimination
  - interleaved with research (forgetting learned clauses = reduce)

- need more general notion of redundancy criteria
  - simply replace “resolvents are tautological” by “resolvents on l are RUP”

\[
(a \lor l) \quad \text{RAT on } l \text{ w.r.t. } \quad (\bar{a} \lor b) \land (l \lor c) \land (\bar{l} \lor b) \quad D
\]

- deletion information is again essential (DRAT)
- now mandatory in the main track of the last two SAT competitions

Propagation Redundant (PR)
“Short Proofs Without New Variables” [HeuleKieslBiere-CADE’17] best paper

- more general than RAT: short proofs for pigeon hole formulas without new variables

- \( C \) propagation redundant if \( \exists \) (partial) assignment \( \omega \) satisfying \( C \) with \( F \mid \bar{C} \vdash_1 F \mid \omega \)