SAT (Modulo Theories) = Resolution

Questions and Challenges

Invited talk, IJCAR 2012 - Manchester

Robert Nieuwenhuis

(+ Ignasi Abío, Albert Oliveras, Enric Rodríguez, Javier Larrosa, ...)

Barcelogic Research Group, Tech. Univ. Catalonia, Barcelona
The objective of this talk is to explain:

- Current SAT and SAT Modulo Theories (SMT) technology.

- Our current aim: extend applications from verification to other industrial combinatorial optimization problems: scheduling, timetabling...

- theoretical limitations

- ways to overcome these limitations

- trade-offs

- challenges
Outline of this talk
Outline of this talk

- Good vs Bad
Outline of this talk

- Good vs Bad in SAT
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is SAT Modulo Theories (SMT)?
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is SAT Modulo Theories (SMT)?
- Lazy approach, improved Lazy approach.
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is SAT Modulo Theories (SMT)?
- Lazy approach, improved Lazy approach.
- \( \text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver} \)
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is SAT Modulo Theories (SMT)?
- Lazy approach, improved Lazy approach.
- \( \text{DPLL}(T) = \text{DPLL}(X) + T\)-Solver
- CP-like theories and \( T\)-solvers. Examples.
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is SAT Modulo Theories (SMT)?
- Lazy approach, improved Lazy approach.
- $DPLL(T) = DPLL(X) + T$-Solver
- CP-like theories and $T$-solvers. Examples.
- Proof complexity and other insights
Outline of this talk

- **Good vs Bad** in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is **SAT Modulo Theories (SMT)**?
- Lazy approach, improved Lazy approach.
- \( \text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver} \)
- CP-like theories and \( T \)-solvers. Examples.
- Proof complexity and other insights
- When can SAT beat SMT? Hybrids!
Outline of this talk

- Good vs Bad in SAT
- SAT Solvers. Why do they work so well? Three basic ideas.
- Same ideas not as successful in general complete CP search?
- What is SAT Modulo Theories (SMT)?
- Lazy approach, improved Lazy approach.
- DPLL($T$) = DPLL($X$) + $T$-Solver
- CP-like theories and $T$-solvers. Examples.
- Proof complexity and other insights
- When can SAT beat SMT? Hybrids!
- The impact of auxiliary variables
Good vs Bad in SAT Solvers

Decades of academic and industrial efforts in SAT
Lots of $$$ from, e.g., EDA (Electronic Design Automation)
Decades of academic and industrial efforts in SAT
Lots of $$$ from, e.g., EDA (Electronic Design Automation)
Lesson: Real-world problems $\neq$ random or artificial ones!
Decades of academic and industrial efforts in SAT
Lots of $$$ from, e.g., EDA (Electronic Design Automation)
Lesson: Real-world problems ≠ random or artificial ones!
What’s GOOD? Complete solvers:
- outperforming by far the other methods (see later why)
- on real-world problems from many sources, with a
  - single, fully automatic, push-button, var selection strategy!
- Hence modeling is essentially declarative.
Good vs Bad in SAT Solvers

Decades of academic and industrial efforts in SAT
Lots of $$\$$ from, e.g., EDA (Electronic Design Automation)

Lesson: Real-world problems $\neq$ random or artificial ones!

What’s GOOD? Complete solvers:
- outperforming by far the other methods (see later why)
- on real-world problems from many sources, with a
- single, fully automatic, push-button, var selection strategy!
- Hence modeling is essentially declarative.

What’s BAD?
- Very low-level language: need modeling and encoding tools
- Sometimes no adequate/compact encodings: arithmetic...
- Answers “unsat” or model. Optimization not as well studied.
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A\parallel F$ (see [NOT], JACM’06):

**Assignment** $A$ : \hspace{1cm} **Clause set** $F$ :

$\emptyset \parallel \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor 6, 6\lor \overline{5}\lor \overline{2} \Rightarrow$
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

**Assignment** $A : \quad$ **Clause set** $F :$

<table>
<thead>
<tr>
<th>0</th>
<th>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</th>
<th>\Rightarrow (Decide)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>\Rightarrow (UnitPropagate)</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form \( A \parallel F \) (see [NOT], JACM’06):

Assignment \( A \) : Clause set \( F \) :

\[
\begin{align*}
\emptyset & \quad \parallel \quad \overline{1} \lor 2, \quad \overline{3} \lor 4, \quad \overline{5} \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad (\text{Decide}) \\
1 & \quad \parallel \quad \overline{1} \lor 2, \quad \overline{3} \lor 4, \quad \overline{5} \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad (\text{UnitPropagate}) \\
1, 2 & \quad \parallel \quad \overline{1} \lor 2, \quad \overline{3} \lor 4, \quad \overline{5} \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad (\text{Decide})
\end{align*}
\]
here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$</th>
<th>Clause set $F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$1 \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$ $\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$1 \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$ $\Rightarrow$ (UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$1 \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$ $\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$1 \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$ $\Rightarrow$ (UnitPropagate)</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$</th>
<th>Clause set $F$</th>
<th>Decide/UnitPropagate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2$</td>
<td>(\Rightarrow) (Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2$</td>
<td>(\Rightarrow) (UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2$</td>
<td>(\Rightarrow) (Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2$</td>
<td>(\Rightarrow) (UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2$</td>
<td>(\Rightarrow) (Decide)</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$</th>
<th>Clause set $F$</th>
<th>Decide/UnitPropagate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1}\lor 2$, $\overline{3}\lor 4$, $\overline{5}\lor \overline{6}$, $6\lor 5\lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1$</td>
<td>$\overline{1}\lor 2$, $\overline{3}\lor 4$, $\overline{5}\lor \overline{6}$, $6\lor 5\lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1\ 2$</td>
<td>$\overline{1}\lor 2$, $\overline{3}\lor 4$, $\overline{5}\lor \overline{6}$, $6\lor 5\lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1\ 2\ 3$</td>
<td>$\overline{1}\lor 2$, $\overline{3}\lor 4$, $\overline{5}\lor \overline{6}$, $6\lor 5\lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1\ 2\ 3\ 4$</td>
<td>$\overline{1}\lor 2$, $\overline{3}\lor 4$, $\overline{5}\lor \overline{6}$, $6\lor 5\lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1\ 2\ 3\ 4\ 5$</td>
<td>$\overline{1}\lor 2$, $\overline{3}\lor 4$, $\overline{5}\lor \overline{6}$, $6\lor 5\lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \Vert F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$</th>
<th>Clause set $F$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 2$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1 2 3$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 2 3 4$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1 2 3 4 5$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 2 3 4 5 6$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td></td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form \( A \parallel F \)  (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment ( A )</th>
<th>Clause set ( F )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (Decide)</td>
</tr>
<tr>
<td>( 1 )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (UnitPropagate)</td>
</tr>
<tr>
<td>( 1 2 )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (Decide)</td>
</tr>
<tr>
<td>( 1 2 3 )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (UnitPropagate)</td>
</tr>
<tr>
<td>( 1 2 3 4 )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (Decide)</td>
</tr>
<tr>
<td>( 1 2 3 4 5 )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (UnitPropagate)</td>
</tr>
<tr>
<td>( 1 2 3 4 5 \overline{6} )</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 )</td>
<td>( \Rightarrow ) (Backtrack)</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$ :</th>
<th>Clause set $F$ :</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (Backtrack)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (Backtrack)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$ $\Rightarrow$ (Backtrack)</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

**Assignment** $A :$ **Clause set** $F :$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$F$</th>
<th>$\implies$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 \ 2$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1 \ 2 \ 3$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 \ 2 \ 3 \ 4$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1 \ 2 \ 3 \ 4 \ 5$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 \ 2 \ 3 \ 4 \ 5 \ \overline{6}$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Backtrack)</td>
</tr>
<tr>
<td>$1 \ 2 \ 3 \ 4 \ \overline{5}$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>model found!</td>
</tr>
</tbody>
</table>
DPLL (or CDCL) SAT Solvers

Here: DPLL (= Davis-Putnam-Loveland-Logemann) = CDCL

An Abstract DPLL state has the form $A \parallel F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$</th>
<th>Clause set $F$</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>Decide</td>
</tr>
<tr>
<td>$1$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>$12$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>Decide</td>
</tr>
<tr>
<td>$123$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>$1234$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>Decide</td>
</tr>
<tr>
<td>$12345$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>$12345\overline{6}$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>Backtrack</td>
</tr>
<tr>
<td>$1234\overline{5}$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td></td>
</tr>
</tbody>
</table>

Model found!

More rules: Backjump, Learn, Forget, Restart [M-S,S,M,...]!
Same example as before. Remember: Backtrack gave 1 2 3 4 5.

But: decision level 3 4 is irrelevant for the conflict 6 ∨ 5 ∨ 2:

\[
\emptyset \quad | \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{6} \lor 5 \lor 2 \implies \text{(Decide)}
\]

1 2 3 4 5 6 \quad | \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{5} \lor 2 \lor 6 \implies \text{(Backjump)}
Backtrack vs. Backjump

Same example as before. Remember: Backtrack gave $1 \ 2 \ 3 \ 4 \ 5$.

But: decision level $3 \ 4$ is irrelevant for the conflict $6 \lor \overline{5} \lor \overline{2}$:

$$
\emptyset \quad \parallel \quad \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)}
$$

$$
1 \ 2 \ 3 \ 4 \ 5 \ 6 \quad \parallel \quad \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Backjump)}
$$

$$
1 \ 2 \ 5 \quad \parallel \quad \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \ldots
$$
Backtrack vs. Backjump

Same example as before. Remember: Backtrack gave 1 2 3 4 5.

But: decision level 3 4 is irrelevant for the conflict $6 \lor \overline{5} \lor \overline{2}$:

\[
\emptyset \quad \quad \mid \quad \overline{1} \lor 2, \quad \overline{3} \lor 4, \quad \overline{5} \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)}
\]

\[
\vdots \quad \vdots \quad \vdots
\]

\[
1 \quad 2 \quad 3 \quad 4 \quad 5 \quad \overline{6} \quad \mid \quad \overline{1} \lor 2, \quad \overline{3} \lor 4, \quad \overline{5} \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Backjump)}
\]

\[
1 \quad 2 \quad \overline{5} \quad \mid \quad \overline{1} \lor 2, \quad \overline{3} \lor 4, \quad \overline{5} \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \ldots
\]

Backjump =

1. Conflict Analysis: “Find” a backjump clause $C \lor l$ (here, $\overline{2} \lor \overline{5}$)
   - that is a logical consequence of $F$
   - that reveals a unit propagation of $l$ at earlier decision level $d$ (i.e., where its part $C$ is false)

2. Return to decision level $d$ and do the propagation.
Conflict Analysis: find backjump clause

Example. Consider assignment: \( \ldots 6 \ldots \bar{7} \ldots 9 \) and let \( F \) contain:
\[
\bar{9} \lor 6 \lor 7 \lor \bar{8}, \quad 8 \lor 7 \lor 5, \quad \bar{6} \lor 8 \lor 4, \quad \bar{4} \lor 1, \quad \bar{4} \lor 5 \lor 2, \quad 5 \lor 7 \lor 3, \quad 1 \lor \bar{2} \lor 3.
\]
UnitPropagate gives \( \ldots 6 \ldots \bar{7} \ldots 9 \ \bar{8} \ \bar{5} \ 4 \ \bar{1} \ 2 \ \bar{3} \). Conflict w/ \( 1 \lor \bar{2} \lor 3! \)

C.An. = do resolutions in reverse order backwards from conflict:
\[
\begin{array}{c}
5 \lor 7 \lor \bar{3} \\
\bar{4} \lor 5 \lor 2 \\
\bar{4} \lor 1 \\
\bar{6} \lor 8 \lor 4 \\
8 \lor 7 \lor \bar{5} \\
8 \lor 7 \lor \bar{6}
\end{array}
\]

until reaching clause with only 1 literal of last decision level.

Can use this backjump clause \( 8 \lor 7 \lor \bar{6} \) for Backjump to \( \ldots 6 \ldots \bar{7} \ 8 \).
Yes, but why is DPLL really that good?

Three key ingredients that only work if used TOGETHER:
Yes, but why is DPLL really that good?

Three key ingredients that only work if used TOGETHER:

1. Learn at each conflict backjump clause as a lemma (“nogood”):
   - makes UnitPropagate more powerful
   - prevents EXP repeated work in future similar conflicts
Yes, but why is DPLL really that good?

Three key ingredients that only work if used TOGETHER:

1. Learn at each conflict backjump clause as a lemma ("nogood"):  
   - makes UnitPropagate more powerful  
   - prevents EXP repeated work in future similar conflicts

2. Decide on variables with many occurrences in recent conflicts:  
   - Dynamic activity-based heuristics (former VSIDS implm.)  
   - idea: work off, one by one, clusters of tightly related vars  
     (try DPLL on two independent instances together...)
Yes, but why is DPLL really that good?

Three key ingredients that only work if used TOGETHER:

1. Learn at each conflict backjump clause as a lemma ("nogood"):  
   - makes UnitPropagate more powerful  
   - prevents EXP repeated work in future similar conflicts

2. Decide on variables with many occurrences in recent conflicts:  
   - Dynamic activity-based heuristics (former VSIDS implm.)  
   - idea: work off, one by one, clusters of tightly related vars (try DPLL on two independent instances together...)

3. Forget from time to time low-activity lemmas:  
   - crucial to keep UnitPropagate fast and memory affordable  
   - idea: lemmas from worked-off clusters no longer needed!
Not the same success doing this in CP...

It’s not easy to get everything together right. But also (I think):
Not the same success doing this in CP...

It’s not easy to get everything **together** right. But also (I think):

- Static (e.g., first-fail) heuristics used
  - effect: work simultaneously on **too unrelated** variables
  - would require storing **too many** nogoods at the same time
It’s not easy to get everything **together** right. But also (I think):

- **Static** (e.g., **first-fail**) heuristics used
  - effect: work simultaneously on **too unrelated** variables
  - would require storing **too many** nogoods at the same time

- **No simple uniform underlying language** (as SAT’s clauses):
  - hard to express nogoods (in SAT, 1st-class citizens: clauses)
  - hard to understand conflict analysis
  - hard to implement things **really** efficiently
Not the same success doing this in CP...

It’s not easy to get everything together right. But also (I think):

- **Static (e.g., first-fail) heuristics used**
  - effect: work simultaneously on **too unrelated** variables
  - would require storing **too many** nogoods at the same time

- **No simple uniform underlying language** (as SAT’s clauses):
  - hard to express nogoods (in SAT, 1st-class citizens: clauses)
  - hard to understand conflict analysis
  - hard to implement things **really** efficiently

- **Learning nogoods not found very useful...**
  - mislead by random/academic pbs?
  - Indeed, it is **useless** isolatedly, and also on **random** pbs!
It’s not easy to get everything \textit{together} right. But also (I think):

- Static (e.g., first-fail) heuristics used
  - effect: work simultaneously on \textit{too unrelated} variables
  - would require storing \textit{too many} nogoods at the same time

- No simple uniform underlying language (as SAT’s clauses):
  - hard to express nogoods (in SAT, 1st-class citizens: clauses)
  - hard to understand conflict analysis
  - hard to implement things \textit{really} efficiently

- Learning nogoods \textit{not found very useful}...
  - \textit{mislead} by random/academic pbs?
  - Indeed, it is \textit{useless} isolatedly, and also on \textit{random} pbs!

- Learning requires \textit{explaining} filtering algs.! [KB’03,05, ...]
Not the same success doing this in CP...

It’s not easy to get everything together right. But also (I think):

- **Static (e.g., first-fail) heuristics used**
  - effect: work simultaneously on too unrelated variables
  - would require storing too many nogoods at the same time

- **No simple uniform underlying language** (as SAT’s clauses):
  - hard to express nogoods (in SAT, 1st-class citizens: clauses)
  - hard to understand conflict analysis
  - hard to implement things really efficiently

- Learning nogoods **not found very useful**...
  - mislead by random/academic pbs?
  - Indeed, it is useless isolatedly, and also on random pbs!

- Learning requires explaining filtering algs.! [KB’03,05, ...]

Towards a solution... see the next slide...
What is SAT Modulo Theories (SMT)?

Origin: Reasoning about equality, arithmetic, data structures such as arrays, etc., in Software/Hardware verification.

What is SMT? Deciding satisfiability of an (existential) SAT formula with atoms over a background theory $T$

Example 1: $T$ is Equality with Uninterpreted Functions (EUF):
3 clauses: $f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d$

Example 2: several (how many?) combined theories:
2 clauses: $A = \text{write}(B, i+1, x), \quad \text{read}(A, j+3) = y \lor f(i-1) \neq f(j+1)$

Typical verification examples, where SMT is method of choice.
Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[
\begin{align*}
\{ f(g(a)) & \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \} \\
1 & \quad 2 & \quad 3 & \quad 4 \\
\end{align*}
\]

1. Send \{ 1 \lor 2, \ 3, \ 4 \} to SAT solver
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[
\begin{align*}
\{ f(g(a)) & \neq f(c) \vee g(a) = d, \quad g(a) = c, \quad c \neq d \} \\
\{ 1 \lor 2, \quad 3, \quad 4 \} 
\end{align*}
\]

1. Send \{ 1 \lor 2, \quad 3, \quad 4 \} to SAT solver

SAT solver returns model [1, 3, 4]
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[
\begin{align*}
&f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \\
&1 \lor 2, \quad 3, \quad 4
\end{align*}
\]

1. Send \(\{1 \lor 2, \quad 3, \quad 4\}\) to SAT solver

SAT solver returns model \([1, \quad 3, \quad 4]\)

Theory solver says \([1, \quad 3, \quad 4]\) is \(T\)-inconsistent
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[
\begin{align*}
f(g(a)) & \neq f(c) \vee g(a) = d, \\
1 & \quad \quad \quad \quad \quad \quad 2
\end{align*}
\]

\[
\begin{align*}
g(a) & = c, \\
3 & \quad \quad \quad \quad \quad \quad 4
\end{align*}
\]

\[
c \neq d
\]

1. Send \(\{ \overline{1} \lor 2, \ 3, \ \overline{4} \}\) to SAT solver

SAT solver returns model \([\overline{1}, \ 3, \ \overline{4}]\)

Theory solver says \([\overline{1}, \ 3, \ \overline{4}]\) is \(T\)-inconsistent

2. Send \(\{ \overline{1} \lor 2, \ 3, \ \overline{4}, \ 1 \lor \overline{3} \lor \overline{4} \}\) to SAT solver
Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[
\begin{align*}
\frac{f(g(a)) \neq f(c)}{\text{1}} \lor \frac{g(a) = d}{\text{2}}, & \quad \frac{g(a) = c}{\text{3}}, \quad \frac{c \neq d}{\text{4}}
\end{align*}
\]

1. Send \{ \overline{1} \lor 2, \overline{3}, \overline{4} \} to SAT solver
   SAT solver returns model \{ \overline{1}, 3, \overline{4} \}
   Theory solver says \{ \overline{1}, 3, \overline{4} \} is \text{T}-inconsistent

2. Send \{ \overline{1} \lor 2, \overline{3}, \overline{4}, 1 \lor \overline{3} \lor 4 \} to SAT solver
   SAT solver returns model \{ 1, 2, 3, \overline{4} \}
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[
\begin{align*}
\frac{f(g(a)) \neq f(c)}{1} & \lor \frac{g(a) = d}{2}, \\
\frac{g(a) = c}{3}, & \frac{c \neq d}{4}
\end{align*}
\]

1. Send \{ \overline{1} \lor 2, 3, \overline{4} \} to SAT solver

   SAT solver returns model \{ \overline{1}, 3, \overline{4} \}

   Theory solver says \{ \overline{1}, 3, \overline{4} \} is \textit{T}-inconsistent

2. Send \{ \overline{1} \lor 2, 3, \overline{4}, 1 \lor \overline{3} \lor 4 \} to SAT solver

   SAT solver returns model \{ 1, 2, 3, \overline{4} \}

   Theory solver says \{ 1, 2, 3, \overline{4} \} is \textit{T}-inconsistent
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[ f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \]

1. Send \( \{ \overline{1} \lor 2, \ 3, \ \overline{4}\} \) to SAT solver

   SAT solver returns model \([\overline{1}, \ 3, \ \overline{4} ]\)

   Theory solver says \([\overline{1}, \ 3, \ \overline{4} ]\) is \(T\)-inconsistent

2. Send \( \{ \overline{1} \lor 2, \ 3, \ \overline{4}, \ 1 \lor \overline{3} \lor 4 \} \) to SAT solver

   SAT solver returns model \([1, \ 2, \ 3, \ \overline{4} ]\)

   Theory solver says \([1, \ 2, \ 3, \ \overline{4} ]\) is \(T\)-inconsistent

3. Send \( \{ \overline{1} \lor 2, \ 3, \ \overline{4}, \ 1 \lor \overline{3} \lor 4, \ \overline{1} \lor 2 \lor \overline{3} \lor 4 \} \) to SAT solver
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

\[ f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \]

1. Send \( \{ \overline{1} \lor 2, \ 3, \ 4 \} \) to SAT solver
   SAT solver returns model \( [\overline{1}, \ 3, \ 4] \)
   Theory solver says \( [\overline{1}, \ 3, \ 4] \) is \( T \)-inconsistent

2. Send \( \{ \overline{1} \lor 2, \ 3, \ 4, \ 1 \lor \overline{3} \lor 4 \} \) to SAT solver
   SAT solver returns model \( [1, \ 2, \ 3, \ 4] \)
   Theory solver says \( [1, \ 2, \ 3, \ 4] \) is \( T \)-inconsistent

3. Send \( \{ \overline{1} \lor 2, \ 3, \ 4, \ 1 \lor \overline{3} \lor 4, \ \overline{1} \lor 2 \lor \overline{3} \lor 4 \} \) to SAT solver
   SAT solver says UNSAT
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models–
- Check $T$-consistency of partial assignment while being built
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models—
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, find an explanation (a small $T$-inconsistent subset of $M$) and add it as a clause
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, find an explanation (a small $T$-inconsistent subset of $M$) and add it as a clause

- Upon a $T$-inconsistency, add clause and restart
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models—
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause—
- Given a $T$-inconsistent assignment $M$, find an explanation (a small $T$-inconsistent subset of $M$) and add it as a clause

- Upon a $T$-inconsistency, add clause and restart—
- Upon a $T$-inconsistency, do conflict analysis of the explanation and Backjump
DPLL\((T)\) approach ('04)  ([NOT], JACM Nov06)

\[
\text{DPLL}(T) = \text{DPLL}(X) \text{ engine } + \ T-\text{Solvers}
\]

- **Modular and flexible:** can plug in any \(T\)-Solvers into the DPLL\((X)\) engine.

- \(T\)-Solvers specialized and fast in Theory Propagation:
  - Propagate input literals that are theory consequences
  - more pruning in improved lazy SMT
  - \(T\)-Solver also guides search, instead of only validating it
  - fully exploited in conflict analysis (non-trivial)
DPLL($T$) Example  (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

\[
\begin{align*}
&f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \\
&\text{1} \quad \text{2} \quad \text{3} \quad \text{4} \\
&\emptyset \parallel \llbracket \text{1} \lor 2, \ 3, \ 4 \rrbracket \Rightarrow \text{(UnitPropagate)}
\end{align*}
\]
DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

\[
\begin{align*}
\begin{array}{l}
    f(g(a)) & \neq f(c) \lor g(a) = d, \\
    g(a) = c, & \\
    c & \neq d
\end{array}
\end{align*}
\]

\[\emptyset \mid \bar{1} \lor 1, 3, 4 \Rightarrow (\text{UnitPropagate})\]

\[3 \mid \bar{1} \lor 2, 3, 4 \Rightarrow (\text{T-Propagate})\]
DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

\[
\begin{align*}
\begin{array}{ccc}
\{g(a) \neq f(c) \lor g(a) = d, g(a) = c, c \neq d\} \\
\hline
\emptyset & \Rightarrow & (\text{UnitPropagate}) \\
3 & \Rightarrow & (T-\text{Propagate}) \\
3 \ 1 & \Rightarrow & (\text{UnitPropagate})
\end{array}
\end{align*}
\]
DPLL($T$) Example  (the same EUF one)

Notation used:  Abstract DPLL Modulo Theories:

\[
\begin{align*}
\left\{ f(g(a)) \neq f(c) \land \left\{ g(a) = d, \quad g(a) = c, \quad c \neq d \right\} \right. \\
\left\{ g(a) = d, \quad g(a) = c, \quad c \neq d \right\} \\
\left\{ f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \right\} \\
\left\{ f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d \right\}
\end{align*}
\]

\[
\begin{align*}
\emptyset & \quad \implies \quad (\text{UnitPropagate}) \\
3 & \quad \implies \quad (\text{T-Propagate}) \\
3 1 & \quad \implies \quad (\text{UnitPropagate}) \\
3 1 2 & \quad \implies \quad (\text{T-Propagate})
\end{align*}
\]
**DPLL\(T\) Example** (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

\[
\begin{align*}
\{ f(g(a)) \neq f(c) \} & \lor \{ g(a) = d \}, & \{ g(a) = c \}, & \{ c \neq d \} \\
\{ 1 \} & & \{ 2 \} & \{ 3 \} & \{ 4 \}
\end{align*}
\]

| \( \emptyset \) | \( \{ 1 \} \lor \{ 2 \}, \{ 3 \}, \{ 4 \} \) | \( \Rightarrow \) | (UnitPropagate) |
| 3  \{ 1 \} | \( \{ 1 \} \lor \{ 2 \}, \{ 3 \}, \{ 4 \} \) | \( \Rightarrow \) | (T-Propagate) |
| 3  1  \{ 1 \} | \( \{ 1 \} \lor \{ 2 \}, \{ 3 \}, \{ 4 \} \) | \( \Rightarrow \) | (UnitPropagate) |
| 3  1  2  \{ 1 \} | \( \{ 1 \} \lor \{ 2 \}, \{ 3 \}, \{ 4 \} \) | \( \Rightarrow \) | (T-Propagate) |
| 3  1  2  4  \{ 1 \} | \( \{ 1 \} \lor \{ 2 \}, \{ 3 \}, \{ 4 \} \) | \( \Rightarrow \) |
**DPLL(\(T\)) Example** (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

\[
\begin{align*}
  f(g(a)) &\neq f(c) \lor g(a)=d, \quad g(a)=c, \quad c \neq d \\
  1 &\quad 2 &\quad 3 &\quad 4
\end{align*}
\]

\[
\begin{align*}
  \emptyset &\quad 1 \lor 2, \quad 3, \quad 4 \quad \Rightarrow \quad \text{(UnitPropagate)} \\
  3 &\quad 1 \lor 2, \quad 3, \quad 4 \quad \Rightarrow \quad \text{(T-Propagate)} \\
  3 1 &\quad 1 \lor 2, \quad 3, \quad 4 \quad \Rightarrow \quad \text{(UnitPropagate)} \\
  3 1 2 &\quad 1 \lor 2, \quad 3, \quad 4 \quad \Rightarrow \quad \text{(T-Propagate)} \\
  3 1 2 4 &\quad 1 \lor 2, \quad 3, \quad 4 \quad \Rightarrow \quad \text{unsat}
\end{align*}
\]

Conflict at decision level zero. No search in this example.
**DPLL(T) Example**  (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

\[
\begin{align*}
  f(g(a)) &\neq f(c) \lor g(a) = d, \\
  g(a) & = c, \\
  c & \neq d
\end{align*}
\]

\[
\begin{align*}
  \emptyset & \quad \| \quad \neg1 \lor 2, \quad 3, \quad \neg4 \quad \Rightarrow \quad (\text{UnitPropagate}) \\
  3 & \quad \| \quad \neg1 \lor 2, \quad 3, \quad \neg4 \quad \Rightarrow \quad (\text{T-Propagate}) \\
  3 1 & \quad \| \quad \neg1 \lor 2, \quad 3, \quad \neg4 \quad \Rightarrow \quad (\text{UnitPropagate}) \\
  3 1 2 & \quad \| \quad \neg1 \lor 2, \quad 3, \quad \neg4 \quad \Rightarrow \quad (\text{T-Propagate}) \\
  3 1 2 4 & \quad \| \quad \neg1 \lor 2, \quad 3, \quad \neg4 \quad \Rightarrow \quad \text{unsat}
\end{align*}
\]

Conflict at decision level zero. No search in this example.

Explanation for last T-Propagate:

\[
2 \land 3 \rightarrow 4 \quad \text{or, equivalently,} \quad \neg2 \lor \neg3 \lor 4
\]

Explanations are *T*-lemmas, i.e., tautologies (valid clauses) in *T*
Conflict analysis in DPLL(T)

Need to do backward resolution with two kinds of clauses:

- UnitPropagate with clause C: resolve with C (as in SAT)
- T-Propagate of lit: resolve with (small) explanation

\[ l_1 \land \ldots \land l_n \rightarrow \text{lit} \] provided by T-Solver
Conflict analysis in DPLL($T$)

Need to do backward resolution with two kinds of clauses:

- **UnitPropagate** with clause $C$: resolve with $C$ (as in SAT)
- **$T$-Propagate** of $lit$: resolve with (small) explanation
  
  $l_1 \land \ldots \land l_n \rightarrow lit$ provided by $T$-Solver

Implemention ideas  (see again [NOT], JACM’06)

- **UnitPropagate**: store pointer to clause $C$, as in SAT solvers
- **$T$-Propagate**: (pre-)compute explanations at each $T$-Propagate?
  
  - usually **better** only on demand, during conflict analysis
  
  then: need to avoid too new $T$-explanations
  
  - typically only one Explain per approx. 250 $T$-Propagates.
  
  - depends on $T$, etc.
What does $\text{DPLL}(T)$ need from $T$-Solver?

- $T$-consistency check of a set of literals $M$, with:
  - Explain of $T$-inconsistency: find \textit{small} $T$-inconsistent subset of $M$
  - Incrementality: if $l$ is added to $M$, check for $M \cup l$ \textit{faster} than reprocessing $M \cup l$ from scratch.

- Theory propagation: find input $T$-consequences of $M$, with:
  - Explain $T$-Propagate of $l$: find \textit{(small)} subset of $M$ that $T$-entails $l$ (needed in conflict analysis).

- Backtrack $n$: undo last $n$ literals added
The **Barcelogic** SMT solver

- **DPLL(X)** = the Barcelogic SAT solver.

- **T-Solvers** for:
  - Congruences (EUF)
  - Integer/Real Difference Logic
  - Linear Integer/Real Arithmetic
  - Arrays
  - ...

- Last few years, main activity on:
  - typical CP filtering algorithms (next)
A DPLL(alldifferent) example

Example:
Quasi-Group Completion (QGC)
Each row and column must contain 1 . . . n.

Good method: 3-D encoding in SAT
where $p_{ijk}$ means “row $i$ col $j$ has value $k$”:

- at least one $k$ per $[i, j]$: clauses like $p_{i1} \lor \ldots \lor p_{ijn}$
- at most one $k$ per $[i, j]$: 2-lit clauses like $\overline{p_{ij1}} \lor \overline{p_{ij2}}$
- same for exactly one $j$ per $[i, k]$ and $i$ per $[j, k]$
- 1 unit clause per filled-in value, e.g., $p_{313}$

In our 5x5 example, DPLL’s UnitPropagate infers no value
but alldifferent does. Which one?
SMT for the theory of \textit{alldifferent}

QGC Example continued:

\textbf{alldifferent} infers that $x, y$ will consume 1, 2 and hence $z = 3$.

\begin{center}
\begin{tabular}{ccc}
$x$ & $y$ & $z$ \\
3 & 4 & \ \\
3 & 4 & 5 \\
4 & 5 & \ \\
5 & & \\
\end{tabular}
\end{center}

Idea:

- Use 3-D encoding + SMT where $T$ is \textit{alldifferent}. As usual in SMT, $T$-solver knows what $p_{ijk}$'s mean.

- From time to time invoke $T$-solver before \textbf{Decide}, but do always cheap SAT stuff first: \textbf{UnitPropagate}, \textbf{Backjump}, etc.

- $T$-solver e.g., incremental filtering [Regin'94] but with Explain: in our example, the literal $p_{133}$ (meaning $z = 3$) is entailed by \{ $\overline{p_{113}}$, $\overline{p_{114}}$, \ldots, $\overline{p_{135}}$ \} (meaning $x \neq 3$, $x \neq 4$, \ldots, $z \neq 5$).
SMT for the theory of \textit{alldifferent}

Get CP with special-purpose global filtering algorithms, learning, backjumping, automatic variable selection heuristics...

Application to real-world professional \textit{round-robin sports} scheduling

Sometimes better results with weaker \textit{alldiff} propagation
Another example: DPLL\textit{\texttt{(cumulative)}}

Plan $N$ tasks. Each has a duration and uses certain finite resources.
Another example: DPLL($\text{cumulative}$)

Plan $N$ tasks. Each has a duration and uses certain finite resources.

**Pure SMT approach**, modeling with variables $s_{t,h}$:

- $s_{t,h}$ means $\text{start}(t) \leq h$ (so $s_{t,h-1} \land s_{t,h}$ means $\text{start}(t) = h$).
- T-solver propagates resource capacities (using filtering algs.)
Another example: DPLL(cumulative)

Plan $N$ tasks. Each has a duration and uses certain finite resources.

Pure SMT approach, modeling with variables $s_{t,h}$:

- $s_{t,h}$ means $\text{start}(t) \leq h$ (so $s_{t,h-1} \land s_{t,h}$ means $\text{start}(t) = h$).
- T-solver propagates resource capacities (using filtering algs.).

Better “hybrid” approach, adding variables $a_{t,h}$:

- $a_{t,h}$ means task $t$ is active at hour $h$
- Time-resource decomposition (AgounBel93, Schutt+09): quadratic no. of clauses like $s_{t,h-\text{duration}(t)} \land s_{t,h} \longrightarrow a_{t,h}$
- T-solver handles, for each hour $h$ and each resource $r$, one Pseudo-Boolean constr. like $3a_{t,h} + 4a_{t',h} + \ldots \leq \text{capacity}(r)$

Very good results.
Another example: DPLL(cumulative)

Plan $N$ tasks. Each has a duration and uses certain finite resources.

**Pure SMT approach**, modeling with variables $s_{t,h}$:

- $s_{t,h}$ means $\text{start}(t) \leq h$ (so $\overline{s_{t,h-1}} \land s_{t,h}$ means $\text{start}(t) = h$).
- T-solver propagates resource capacities (using filtering algs.)

**Better “hybrid” approach**, adding variables $a_{t,h}$:

- $a_{t,h}$ means task $t$ is active at hour $h$
- Time-resource decomposition (AgounBel93, Schutt+09): quadratic no. of clauses like $\overline{s_{t,h-\text{duration}(t)}} \land s_{t,h} \longrightarrow a_{t,h}$
- T-solver handles, for each hour $h$ and each resource $r$, one Pseudo-Boolean constr. like $3a_{t,h} + 4a_{t',h} + \ldots \leq \text{capacity}(r)$

Very good results.
But... why can SAT sometimes still beat SMT? See below!
The pigeon-hole principle for $n$ pigeons and $n − 1$ holes:

Let $PHP_{n−1}^n$ denote the set of clauses:

- $x_{i,1} ∨ \ldots ∨ x_{i,n−1}$ for $i = 1 \ldots n$
  (every pigeon is in at least one hole)

- $\overline{x_{i,k}} ∨ \overline{x_{j,k}}$ for $1 ≤ i < j ≤ n$ and $1 ≤ k ≤ n − 1$
  (no two pigeons are in the same hole)

[Haken’85]:
Any resolution refutation of $PHP_{n−1}^n$ requires size exponential in $n$.

**Note:** pigeon-hole-like situations do occur in practice. E.g., hidden in scheduling/timetabling: $n−1$ (human) resources for $n$ tasks...
Proof complexity and other insights (II)

[Zhang&Malik’03]: CDCL SAT solvers can generate a proof trace file, from which one can extract, for each lemma, a resolution proof from input clauses:

\[
\begin{align*}
\text{id}_2: & \quad 5 \lor 7 \lor \overline{3} \\
\text{id}_1: & \quad 1 \lor 2 \lor 3 \\
\text{id}_3: & \quad 5 \lor 7 \lor 1 \lor 2 \\
\text{id}_k: & \quad \ldots \\
\text{id}: & \quad \vdots
\end{align*}
\]

One trace line per conflict/lemma: \( \text{id} \leftarrow \{\text{id}_1 \ldots \text{id}_k\} \)

If input is unsat, conflict at DL zero: last lemma is the empty clause:

\[
\text{trace file} \geq (\text{binary}) \text{ resolution refutation:}
\]

\[
\text{SAT solver runtime} \geq \text{size of smallest resolution refutation.}
\]
SMT solvers can also generate such traces.

SMT unsat proofs are modular, with two parts:

- A (purely propositional) resolution refutation from:
  - the clauses of the input CNF
  - the generated explanations
    (these clauses are written in the trace as well)
- For each explanation clause, an independent proof in (its) $T$.

So, after all, SMT does generate a SAT encoding, but lazily.

SMT solver runtime $\geq$ size of smallest resolution refutation.
In which cases can SAT beat SMT?

- SMT’s *lazy* SAT encoding could end up being a *full* one
- And... this full encoding could be a rather *naive* one!

**Example:**

\[ T = \text{cardinality constraint } \sum_{i=1}^{n} x_i \leq k. \]

*T*-solver is just a counter.

Input: propositional clauses implying \( \sum_{i=1}^{n} x_i > k. \)

Refutation requires all \({\binom{n}{k+1}}\) explanations of the form

\[ y_1 \land \ldots \land y_k \rightarrow \overline{y} \]
In which cases can SAT beat SMT?

- SMT’s lazy SAT encoding could end up being a full one
- And... this full encoding could be a rather naive one!

Example:

\[ T = \text{cardinality constraint } x_1 + \ldots + x_n \leq k. \]

\( T \)-solver is just a counter.

Input: propositional clauses implying \( x_1 + \ldots + x_n > k. \)

Refutation requires all \( \binom{n}{k+1} \) explanations of the form

\[ y_1 \land \ldots \land y_k \rightarrow \overline{y} \]

For \( T \)-constraints triggering many explanations, i.e., bottle necks, better use good SAT encoding with auxiliary variables!

Here, e.g., Cardinality Networks: \( O(n \log^2 k) \) clauses and aux. vars.
When to use SAT, and when SMT?

- Most constraints are no bottle necks and generate very few explanations $\implies$ handle with SMT.
- For bottle necks, better use SAT encoding with aux vars

Little detail.... problems have many constraints, and cannot predict at encoding time which one will be a bottle neck!
When to use SAT, and when SMT?

- Most constraints are no bottle necks and generate very few explanations $\implies$ handle with SMT.
- For bottle necks, better use SAT encoding with aux vars

Little detail.... problems have many constraints, and cannot predict at encoding time which one will be a bottle neck!

Solution from [Abío and Stuckey, CP 2012]:

- Start with SMT, but generate SAT encoding with aux vars on the fly for those constraint (parts) appearing in many conflicts
- Usually improves best of SAT/SMT, and never really worse.
When to use SAT, and when SMT?

- Most constraints are no bottle necks and generate very few explanations $\implies$ handle with SMT.
- For bottle necks, better use SAT encoding with aux vars

Little detail.... problems have many constraints, and cannot predict at encoding time which one will be a bottle neck!

Solution from [Abío and Stuckey, CP 2012]:
- Start with SMT, but generate SAT encoding with aux vars on the fly for those constraint (parts) appearing in many conflicts
- Usually improves best of SAT/SMT, and never really worse.

Challenges/questions:
- Generalize beyond cardinality and pseudo-boolean constraints
- Whether/how SAT solver should split (decide) on aux vars?
Extended Resolution (ER) introduces auxiliary vars (definitions).
No problem family found (yet?) without short ER unsat proofs.
Auxiliary variables & proof complexity

Extended Resolution (ER) introduces auxiliary vars (definitions). No problem family found (yet?) without short ER unsat proofs.

Two ways to (try to) exploit this:

- **Use encoding with aux. vars. and split/decide on them?**
  Compatible with [AS’12] on-the-fly SMT → SAT encodings.
  Limitation: No P-size domain-consistent SAT encoding, not even with aux vars, for, e.g., alldiff [BessiereEtal’09].

- **CDCL SAT solvers that introduce aux var definitions**
  [AudemardKS’10,Huang’10]
Auxiliary variables & proof complexity

Extended Resolution (ER) introduces auxiliary vars (definitions). No problem family found (yet?) without short ER unsat proofs.

Two ways to (try to) exploit this:

- **Use encoding with aux. vars. and split/decide on them?**
  Compatible with [AS’12] on-the-fly SMT → SAT encodings.
  Limitation: No P-size domain-consistent SAT encoding, not even with aux vars, for, e.g., alldiff [BessiereEtal’09].

- **CDCL SAT solvers that introduce aux var definitions** [AudemardKS’10,Huang’10]

Also, between resolution and extended resolution: cutting planes.
DPLL-like linear integer arithmetic solvers like [JdM’11]
Concluding remarks

Apart from the challenges we have mentioned...

- Need more CP filtering algorithms with explain.

- Progress (but need more) in optimization problems:
  - Branch and bound is just another SMT theory [SAT’06]
  - Framework for branch and bound w/ lower bounding and optimality proof certificates [SAT’09, JAR’11].
  - MAX-SMT.

- ...
Concluding remarks

Apart from the challenges we have mentioned...

- Need more CP filtering algorithms with explain.

- Progress (but need more) in optimization problems:
  - Branch and bound is just another SMT theory [SAT’06]
  - Framework for branch and bound w/ lower bounding and optimality proof certificates [SAT’09, JAR’11].
  - MAX-SMT.

- ...

Barcelogic looks for industrial problems, partners, (EU) projects...

Thank You!