SAT Modulo Theories:
Can we get the best of two worlds?

Invited talk, CP 2010 - St Andrews

Robert Nieuwenhuis
(+ Albert Oliveras, Enric Rodríguez, Roberto Asín, Javier Larrosa, ...)

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

- What SAT Modulo Theories (SMT) is.

Our current aim: bring SMT from verification applications to other more typical CP ones: scheduling, timetabling...

Can we use SMT trying to get the best of two worlds?:

- From SAT: efficiency, robustness, no need for tuning.

- From general complete methods in CP (note: CP ⊃ SAT): expressiveness, rich modeling languages, special-purpose algorithms for arithmetic, for global constraints....
Outline of this talk
Outline of this talk

- Good vs Bad
Outline of this talk

- Good vs Bad in SAT and other complete CP search methods.
Outline of this talk

- Good vs Bad in SAT and other complete CP search methods.
- SAT Solvers. Why do they work so well? Three basic ideas.
Outline of this talk

- Good vs Bad in SAT and other complete CP search methods.
- 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 and other complete CP search methods.
- 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 and other complete CP search methods.
- 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 and other complete CP search methods.
- 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.
- Our DPLL($T$) approach: $DPLL(T) = DPLL(X) + T$-Solver.
Outline of this talk

- Good vs Bad in SAT and other complete CP search methods.
- 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.
- Our DPLL$(T)$ approach: $DPLL(T) = DPLL(X) + T$-Solver.
- The Barcelogic SMT solver. Theories and $T$-Solvers
Outline of this talk

- Good vs Bad in SAT and other complete CP search methods.
- 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.
- Our DPLL\((T)\) approach: \(\text{DPLL}(T) = \text{DPLL}(X) + T\)-Solver.
- The Barcelogic SMT solver. Theories and \(T\)-Solvers
- CP-like theories and \(T\)-solvers. Examples.
Outline of this talk

- Good vs Bad in SAT and other complete CP search methods.
- 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.
- Our $DPLL(T)$ approach: $DPLL(T) = DPLL(X) + T$-Solver.
- The Barcelogic SMT solver. Theories and $T$-Solvers
- CP-like theories and $T$-solvers. Examples.
- Proof complexity and other insights

Barcelogic - Tech. Univ. Catalonia (UPC)
Outline of this talk

- **Good vs Bad** in SAT and other complete CP search methods.
- 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.
- Our **DPLL(T)** approach: $\text{DPLL}(T) = \text{DPLL}(X) + T$-Solver.
- The **Barcelogic** SMT solver. Theories and **T-Solvers**
- CP-like theories and $T$-solvers. Examples.
- Proof complexity and other insights
- Concluding remarks
What is meant by CP solver in this talk?

“Typical” state-of-the-art solver with:

- complete systematic search
- backtracking (no backjumping)
- no learning
- rich modeling languages
- sophisticated:
  - heuristics for branching variable selection (e.g., first-fail)
  - heuristics for branching value selection
  - special-purpose global constraint propagation algorithms

NB: for some problems, complete CP/SAT/SMT all inadequate!
Decades of academic and industrial efforts in SAT
Lots of $$$ from, e.g., EDA (Electronic Design Automation)
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!
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.
Decades of academic and industrial efforts in SAT
Lots of $$$ from, e.g., EDA (Electronic Design Automation)

Lesson: Real-world problems \( \not= \) 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.

++ SAT in CP’10 Procs! E.g., pg 398, Petke&Jeavons’ abstract ends: “We (...) show that, without being explicitly designed to do so, current clause-learning SAT solvers efficiently simulate \( k \)-consistency techniques, for all values of \( k \) [and] (...) efficiently solve certain families of CSP instances which are challenging for conventional CP solvers”.

Barcelogic - Tech. Univ. Catalonia (UPC)
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 ≠ 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
- no good encodings for many aspects: **arithmetic...**
- Answers “unsat” or model. **Optimization** not as well studied.
Good vs Bad in general CP Solvers
Good vs Bad in general CP Solvers

What’s GOOD?

- Expressive modeling constructs and languages
- Specialized algorithms for many (global) constraints
- Optimization aspects better studied
Good vs Bad in general CP Solvers

What’s GOOD?

- Expressive modeling constructs and languages
- Specialized algorithms for many (global) constraints
- Optimization aspects better studied

What’s BAD or, well, not so good?

- Performance(?)
- Not quite automatic or push-button
  Heuristics tuning per problem (or even per instance)
- In CP Procs, sometimes only “academic” experiments:
  – on random or artificial problems (sometimes not realistic)
  – no big database of real-world/industrial instances
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 $A :$ Clause set $F :$

$\emptyset \parallel \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2 \Rightarrow$
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 $A$ : Clause set $F$ :

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

$1$ $\parallel \overline{1} \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (\text{UnitPropagate})$
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** \( F : \)

\[
egin{array}{c|c|c}
\emptyset & 1 \lor 2, \quad 3 \lor 4, \quad 5 \lor 6, \quad 6 \lor 5 \lor 2 & \Rightarrow \text{(Decide)} \\
1 & 1 \lor 2, \quad 3 \lor 4, \quad 5 \lor 6, \quad 6 \lor 5 \lor 2 & \Rightarrow \text{(UnitPropagate)} \\
1 2 & 1 \lor 2, \quad 3 \lor 4, \quad 5 \lor 6, \quad 6 \lor 5 \lor 2 & \Rightarrow \text{(Decide)} \\
\end{array}
\]
## 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>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>$1 \ 2$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>$1 \ 2 \ 3$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor \overline{2}$</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 $A \parallel F$ (see [NOT], JACM’06):

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

Barcelologic - Tech. Univ. Catalonia (UPC)
# 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>Action</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 \overline{2} )</td>
<td>(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 \overline{2} )</td>
<td>(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 \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 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 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 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 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):

**Assignment** $A : \quad \text{Clause set } F :$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$F$</th>
<th>Action</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>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>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Backtrack)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(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>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>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>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>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>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>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>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>Backtrack</td>
</tr>
<tr>
<td>$1 2 3 4 \overline{5}$</td>
<td>$\overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor \overline{5} \lor 2$</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 $A \| F$ (see [NOT], JACM’06):

<table>
<thead>
<tr>
<th>Assignment $A$</th>
<th>Clause set $F$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>Decide</td>
</tr>
<tr>
<td>1</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>1 2</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>Decide</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>Decide</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>1 2 3 4 5 $\bar{6}$</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>Backtrack</td>
</tr>
<tr>
<td>1 2 3 4 $\bar{5}$</td>
<td>$\bar{1} \lor 2$, $\bar{3} \lor 4$, $\bar{5} \lor \bar{6}$, $6 \lor \bar{5} \lor 2$</td>
<td>model found!</td>
</tr>
</tbody>
</table>

More rules: Backjump, Learn, Forget, Restart [M-S,S,M,...]!
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 ∨ 5 ∨ 2:

∅ || 1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (Decide)

1 2 3 4 5 6 || 1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (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 ∨ 5 ∨ 2:

∅  ∩  1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2  ⇒  (Decide)

1 2 3 4 5 6  ∩  1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2  ⇒  (Backjump)
1 2 5  ∩  1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2  ⇒  …
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 $6 \lor \bar{5} \lor \bar{2}$:

$$\emptyset \mid \bar{1} \lor 2, \ ar{3} \lor 4, \ \bar{5} \lor 6, \ 6 \lor 5 \lor \bar{2} \Rightarrow \text{(Decide)}$$

But:

$$1 \ 2 \ 3 \ 4 \ 5 \ 6 \mid \bar{1} \lor 2, \ ar{3} \lor 4, \ \bar{5} \lor 6, \ 6 \lor 5 \lor 2 \Rightarrow \text{(Backjump)}$$

$$1 \ 2 \ 5 \mid \bar{1} \lor 2, \ ar{3} \lor 4, \ \bar{5} \lor 6, \ 6 \lor 5 \lor \bar{2} \Rightarrow \ldots$$

Backjump =

1. **Conflict Analysis**: “Find” a backjump clause $C \lor l$ (here, $\bar{2} \lor \bar{5}$)
   - 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: \(\ldots 6\ldots \overline{7}\ldots 9\) and let \(F\) contain:
\[
\overline{9} \lor \overline{6} \lor \overline{7} \lor 8, \ 8 \lor 7 \lor 5, \ \overline{6} \lor 8 \lor 4, \ \overline{4} \lor 1, \ \overline{4} \lor 5 \lor 2, \ 5 \lor 7 \lor 3, \ 1 \lor 2 \lor 3.
\]
UnitPropagate gives \(\ldots 6\ldots \overline{7}\ldots 9 \ 8 \ 5 \ 4 \ 1 \ 2 \ 3\). Conflict w/ \(1 \lor 2 \lor 3\)!

C.An. = do resolutions in reverse order backwards from conflict:

\[
\begin{align*}
&\overline{4} \lor \overline{5} \lor \overline{2} & \overline{4} \lor 1 \lor 7 \\
&\overline{4} \lor 1 \lor 2 & \overline{4} \lor 5 \lor 7 \lor 1 \\
&\overline{6} \lor 8 \lor 4 & 5 \lor 7 \lor 4 \\
&8 \lor 7 \lor \overline{5} & \overline{6} \lor 8 \lor 7 \lor 5 \\
&8 \lor 7 \lor \overline{6}
\end{align*}
\]

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

Can use this backjump clause \(8 \lor 7 \lor \overline{6}\) for Backjump to \(\ldots 6\ldots \overline{7} \ 8\).
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...)

---

**Barcelogic** - Tech. Univ. Catalonia (UPC)
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
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
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 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, ...]
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.
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

1. Send \{1 \lor 2, 3, 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:

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

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

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

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

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

2. Send \{ 1 \lor 2, 3, 4, 1 \lor 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 \( \{ \overline{1} \lor 2, \ 3, \ \overline{4} \} \) 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 \( \{ \overline{1} \lor 2, \ 3, \ \overline{4}, \ 1 \lor 3 \lor 4 \} \) 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:

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

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
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

1. Send \( \{\top \lor 2, \; 3, \; \bot \} \) to SAT solver
   
   SAT solver returns model \( \{\top, \; 3, \; \bot\} \)
   
   Theory solver says \( \{\top, \; 3, \; \bot\} \) is \( T \)-inconsistent

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

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

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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 T-inconsistent

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

Barcelogic - Tech. Univ. Catalonia (UPC)
DPLL($T$) approach ('04)  ([NOT], JACM Nov06)

DPLL($T$) = DPLL(X) engine + $T$-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($T$) approach is being quite widely adopted (cf. Google).
DPLL(\(T\)) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

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

\[
\emptyset \parallel \bar{1} \lor 2, \ 3, \ \bar{4} \Rightarrow \text{(UnitPropagate)}
\]
Notation used: Abstract DPLL Modulo Theories:

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

\[ \emptyset \quad \Rightarrow \quad 1 \lor 2, 3, 4 \quad \Rightarrow \quad (\text{UnitPropagate}) \]

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

Notation used: Abstract DPLL Modulo Theories:

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

\[
\begin{align*}
\emptyset & \lor \bar{1} \lor 2, 3, 4 \Rightarrow \text{(UnitPropagate)} \\
3 & \lor \bar{1} \lor 2, 3, 4 \Rightarrow \text{(T-Propagate)} \\
3 \ 1 & \lor \bar{1} \lor 2, 3, 4 \Rightarrow \text{(UnitPropagate)}
\end{align*}
\]
Notation used: Abstract DPLL Modulo Theories:

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

\[
\begin{align*}
\emptyset & \quad \models \overline{1} \lor 2, 3, 4 \quad \Rightarrow \quad \text{(UnitPropagate)} \\
3 & \quad \models \overline{1} \lor 2, 3, 4 \quad \Rightarrow \quad \text{(T-Propagate)} \\
3 1 & \quad \models \overline{1} \lor 2, 3, 4 \quad \Rightarrow \quad \text{(UnitPropagate)} \\
3 1 2 & \quad \models \overline{1} \lor 2, 3, 4 \quad \Rightarrow \quad \text{(T-Propagate)}
\end{align*}
\]
DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

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

Notation used: Abstract DPLL Modulo Theories:

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

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

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

Notation used: Abstract DPLL Modulo Theories:

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

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

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

Explanation for last T-Propagate:

\[2 \land 3 \rightarrow 4\]

or, equivalently, \[\bar{2} \lor \bar{3} \lor 4\]

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
  
  \[ l_1 \land \ldots \land l_n \rightarrow lit \]  
  provided by $T$-Solver

Too new $T$-explanations are forbidden!

How should it be implemented? (see again [NOT], JACM’06)

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

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

- Theory propagation: find input \( T \)-consequences of \( M \), with:
  - Explain \( T \)-Propagate of \( l \): find (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) is a state-of-the-art DPLL-based SAT engine: the Barcelogic SAT solver.

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

- New: typical CP filtering algorithms (next)
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 \(p_{ij1} \lor \ldots \lor p_{ijn}\)
- 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 \texttt{alldifferent} does. Which one?
QGC Example continued:

**alldifferent** infers that \(x, y\) will consume 1, 2 and hence \(z = 3\).

Idea:

- Use 3-D encoding + SMT where \(T\) is **alldifferent**. As usual in SMT, \(T\)-solver knows what \(p_{ijk}\)'s mean.
- From time to time invoke \(T\)-solver before Decide, but do always cheap SAT stuff first: UnitPropagate, 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(*cumulative*)

*N* tasks. Each one has a **duration** and uses certain **finite resources**.

**Pure SMT approach**, modeling with variables *s*<sub>*t,h*</sub>:

- *s*<sub>*t,h*</sub> means *start*(t) ≤ *h* (so *s*<sub>*t,h−1*</sub> ∧ *s*<sub>*t,h*</sub> means *start*(t) = *h*).
- **T-solver** propagates resource capacities (using filtering algs.)

**Better “hybrid” approach**, adding variables *a*<sub>*t,h*</sub>:

- *a*<sub>*t,h*</sub> means task *t* is active at hour *h*
- Time-resource decomposition (*AgounBel93*, *Schutt+09*): quadratic no. of clauses like *s*<sub>*t,h−duration(t)* ∧ *s*<sub>*t,h*</sub> → *a*<sub>*t,h*</sub>
- **T-solver** handles, for each hour *h* and each resource *r*, one Pseudo-Boolean constr. like 3*a*<sub>*t,h*</sub> + 4*a*<sub>*t′,h*</sub> + ... ≤ capacity(*r*)

**Very good results.**

Why can SAT sometimes beat SMT? See below.
Proof complexity and other insights

SMT solvers can generate unsat proofs, which come in two parts:

- A resolution refutation from:
  - the clauses of the input CNF
  - the generated explanations (clauses)
- For each explanation clause, an independent proof in (its) $T$.

So, after all, SMT generates a SAT encoding, but lazily.

SMT solver runtime $\geq$ size of smallest resolution proof.
How could SAT beat SMT?

In “artificial-like” problems:

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

Example: \( T = \) cardinality constraints. \( T \)-solver is just a counter.

Unsat instance: \( x_1 + \ldots + x_n \geq k \) and \( x_1 + \ldots + x_n < k \)

Refutation requires all \( \binom{n}{k+1} \) explanations like, e.g.,

\[
x_1 \land \ldots \land x_k \rightarrow \overline{x_{k+1}}
\]

Here a good SAT encoding with auxiliary vars works better.
Splitting on aux vars can give expon. speedup: Extended Resol.

But... some constraints admit no P-size domain-consistent SAT encoding, e.g., alldiff [BessiereEtal’09].
Comparison with Lazy Clause Generation

LCG [OhrimenkoStuckeyCodish07] was the instance of SMT where:

- each time the T-solver detects that lit can be propagated, it generates and adds (forever) the explanation clause, so the SAT-solver can UnitPropagate lit with it.

But as we have seen in this talk, it is usually better to:

- Generate explanations only when needed: at conflict an. time.
- Never add explanations as clauses. Otherwise: die keeping too many explanations (or the whole SAT encoding).

Remember: Forget of the usual lemmas is already Crucial to keep UnitPropagate fast and memory affordable!

Since recently, with these improvements, LCG = SMT.
Concluding remarks

- Need more work on further 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).
  - MAX-SMT.
Concluding remarks

- Need more work on further 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).
  – MAX-SMT.

- Barcelologic is looking for industrial problems, partners, projects (e.g., EU)...

- Thank You!